AFIT/GCS/ENG/93-07

AD-A274 088

WEAPON SYSTEM SENSOR INTEGRATION FOR A DIS-COMPATIBLE VIRTUAL COCKPIT

DTIC
ELECTE
DEC 23 1993
S E D

THESIS

Matthew Nick Erichsen
Captain,USAF

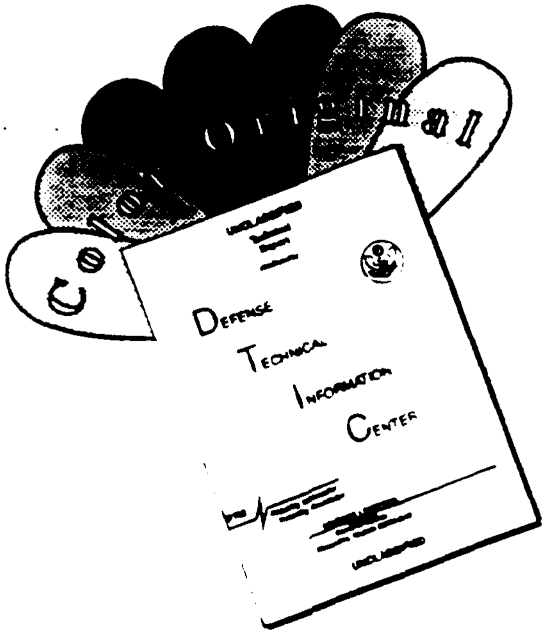
AFIT/GCS/ENG/93-07

Approved for public release; distribution unlimited

93-30927

93 12 22 040

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF COLOR PAGES WHICH DO NOT REPRODUCE LEGIBLY ON BLACK AND WHITE MICROFICHE.

AFIT/GCS/ENG/93-07

WEAPON SYSTEM SENSOR INTEGRATION FOR A DIS-COMPATIBLE VIRTUAL COCKPIT

THESIS

F nted to the Faculty of the Graduate School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science (Computer Science)

DTIC
COPY
INSPECTED
6

Accesion For
NTIS CRA&I ☒
DTIC TAB ☐
Unannounced ☐
Justification ....
By ....
Distribution /
Availability Codes

Dist	Avail and / or Special
A-1	

Matthew Nick Erichsen, B.S.
Captain,USAF

December, 1993

Approved for public release; distribution unlimited

Acknowledgements

Thanks to my advisor LtCol Phil Amburn for his guidance and encouragement which led me to do more than I ever envisioned. My appreciation goes to our ARPA sponsor, LtCol David Neyland, whose invaluable support and vision made this effort a success. I have to thank the "graphics gang" for the mutual support that kept our six clear of any disasters: Captains Andrea Kunz, Mark Snyder, Bill Gerhard, Kirk Wilson, Brian Soltz, and Mike Gardner. Friendships are born through adversity, so I'm sure we are destined to be pals for a life-time.

My deepest appreciation goes to my wife Paula; thank you for holding the household together during the long hours I was away and the encouragement to be a husband and father to those most dear to me. Finally, our love to our friends at Hillside Chapel whose prayers, accountability, and support will be missed. If we don't meet sooner, we'll see you at the call.

Matthew Nick Erichsen

Table of Contents

Page

Acknowledgements ii

List of Figures viii

List of Tables x

Abstract xi

I. Problem Statement 1

1.1 Introduction 1

1.2 Background 3

1.3 Problem Components 4

1.3.1 Realistic Frame Rates 5

1.3.2 Realistic Scenarios 5

1.3.3 Cost of Expanding Capabilities 6

1.4 Approach/Methodology 6

1.4.1 The Radar Display 6

1.4.2 Target Identification 8

1.4.3 FLIR Requirements 9

1.4.4 Weapon System Interface 9

1.4.5 WGS-84 Changes 10

1.5 Conclusions 10

II. Literature Review 11

2.1 Introduction 11

2.2 Flight Simulation 11

Page
2.3 Virtual Environments 12
2.3.1 Projection Mechanisms 13
2.3.2 Immersion Effectiveness 13
2.3.3 Visualization Effectiveness 14
2.4 Current Technology 15
2.5 New Trends 17
2.6 Network Interaction 18
2.7 Conclusion 19

III. Design Decisions 20
3.1 Introduction 20
3.2 Analysis 22
3.2.1 Situational Awareness 24
3.3 Minimum Radar Task Set 27
3.3.1 Navigation 27
3.3.2 Targeting 27
3.4 Minimum FLIR Functions 30

IV. Implementation Process 32
4.1 Performer Interface 33
4.1.1 ObjectSim Application 34
4.1.2 Shared Memory 36
4.2 Scaled Text 37
4.2.1 GL Library standard 37
4.2.2 Wright Lab 2D text (Graphtext/Graphfont) . . . 37
4.3 Arbitrating Input Devices 38
4.4 MFD Object 39
4.4.1 Radar Object 40

Page

4.4.2 FLIR Object 45

4.5 Cursor Object 45

4.5.1 User Interface 45

4.5.2 Accuracy 46

4.5.3 Cursor Designation 47

4.6 INS Object 48

4.7 INS Calculations 48

4.7.1 Radar Screen Coordinates 49

4.7.2 Air Target Computations 52

4.7.3 Ground Target Computations 55

4.7.4 Network Management 56

V. Development Aids And Simulation Environment Enhancements . . . 59

5.1 Moving parts 59

5.2 Drones and Ducks 60

5.3 Timers and Reconstruction 60

5.4 Mini Dart 61

5.5 Round Earth Terrain Model Development 62

VI. Implementing Round Earth 64

6.1 Introduction 64

6.2 Coordinate Systems/Reference Frames 64

6.2.1 GL/Performer Reference Frames 65

6.2.2 WGS 84/Earth-Centered Earth Fixed (ECEF) Frame 69

6.2.3 Earth-Centered Inertial (ECI) Frame 69

6.2.4 Navigation Frame (*n*-frame) 70

6.2.5 Aerodynamic Body Coordinates (*b*-frame) 71

6.3 Coordinate Transformations 71

Page
6.3.1 Direction Cosine Matrix (DCM) 72
6.3.2 Euler Angles 72
6.4 Round Earth/Aerodynamic Model Problems 75
6.5 Round Earth/Performer Problems 76
6.6 The Round Earth Utilities 78
6.6.1 Moving The WGS 84 Area of Interest 78
6.6.2 Public Methods For Round Earth Utilities 83

VII. Results 84
7.1 Performance Limitations 86
7.2 Software Reliability 87
7.3 Problems and Suggestions For Improvements 87
7.3.1 Floating Point Accuracy 87
7.3.2 Mission Duration/Distance Travelled 89
7.3.3 Radar Enhancements 90
7.3.4 FLIR Enhancements 91
7.3.5 Initial VC Orientation 91
7.3.6 Arbitrating Damage In Distributed Simulations . 91
7.3.7 VC Displays And Capabilities 92
7.4 Conclusions 92

Appendix A. Virtual Cockpit User Guide 98
A.1 Introduction 99
A.2 HOTAS Switches And Operation 100
A.2.1 Throttle Switches Radar Submode 100
A.2.2 Throttle Switches FLIR Submode 102
A.2.3 Stick Switches All Modes 103
A.3 Running the Virtual Cockpit 104

Page

A.3.1 Start Up 104

A.3.2 MFD Description 106

A.3.3 HUD Description 108

A.3.4 Gotchas 108

Appendix B. Round Earth Utilities 111

B.1 Header File 112

B.2 Body File 118

Bibliography 136

Vita 140

List of Figures

Figure Page

1. Simplified F-16 Air-To-Air Radar Display 7
2. Radar Object Symbols 8
3. Virtual Cockpit Object Diagram Of Thesis Software 21
4. SAFOR Object Diagram 22
5. Example Avionics System Structure (1) 23
6. Ballistic Bomb Fall Diagram 26
7. Aspect Angle As Angle Off The Tail (AOT) 30
8. ObjectSim High Level Diagram (37:8) 34
9. ObjectSim Typical Network Interface (37:38) 35
10. ObjectSim and Virtual Cockpit Application Data Flow 35
11. HOTAS 39
12. ObjectSim Playerlist 41
13. Sample INS To Radar Conversion 43
14. Cursor Showing Surrounding Basket 47
15. Performer Version 1.2 Simple Frustum 49
16. Logic For Line Of Sight Testing 51
17. Transforming World Coordinates To Radar Screen Coordinates . . . 53
18. miniDART External View 61
19. miniDART Projection Surfaces 62
20. DTED Terrain Transformation Routine Results 63
21. Performer Coordinate System 65
22. Performer Body Coordinate System 66
23. Graphics Library Screen Coordinates 67
24. GL Frustum Coordinate System 68
25. WGS-84 Earth Centered Inertial Coordinate System 69

Figure Page

26. ENV Coordinate System 70

27. Aerodynamic Coordinate System 71

28. Euler Angle Body Axis Rotations 74

29. Aircraft Locally Level But Accelerating "Down". 76

30. Terrain Rotation To -Y Longitude 78

31. Terrain Rotation To Vertical Axis 79

32. Terrain Translation To Earths "Core" 79

33. ALPHA Angular Determination 80

34. BETA Angular Determination 81

35. VC 1.0 Instrument Display 94

36. VC 2.0 Instrument Display Radar Ground Mode 94

37. VC 2.0 Radar Navigation Mode 95

38. VC 2.0 Radar Air Mode With Target Designated 95

39. VC 2.0 FLIR Display With Air Target 96

40. VC 2.0 Radar Target With Radar Missile Launched And Tracking . 96

41. VC 2.0 Moving Parts Extended 97

42. F-15E Forward Panel Front Cockpit 97

43. Basic Dual Throttle Switches 99

44. Basic Stick Switches 99

45. Radar Submode Throttle Switches 100

46. FLIR Submode Throttle Switches 102

47. VC Named Layout 104

48. VC Layout 105

49. Sample MFD Display 110

50. Sample HUD Displays 110

List of Tables

Table Page

1. Comparison of Immersion Issues. 16
2. Comparison of Visualization Issues. 16
3. Reference Frames. 66
4. VC capabilities and future modifications. 93

AFIT/GCS/ENG/93-07

Abstract

This thesis continues the Virtual Cockpit (VC) research which investigates distributed interactive virtual flying environments. The VC v1.0 used the SIMNET protocols to only communicate position and orientation over a common network. A simple cockpit instrumentation configuration and limited head-up display (HUD) showed aircraft state. With VC v1.0 weapons or sensors could not interact in the simulation environment. The VC v2.0 transitions from the SIMNET protocol to a partial implementation of the Distributed Interactive Simulation (DIS) v2.0.3 protocol. Simulated radar and forward looking infra-red (FLIR) sensors were developed to aid operator detection and designation when employing various munition types. Simulated munition types include: radar or IR missiles, free-fall, laser guided, or electro-optic (EO) guided bombs, and a 20mm cannon. Virtual environments were created with CRT out-the-window presentations, color NTSC and monochrome high-resolution helmet mounted displays employing Polhemus head tracking sensors, and simultaneously five-channels on BARCO projectors. Target graphics systems included SGI workstations with Onyx processors using Reality Engines. Graphics rendering was accomplished with an AFIT developed object oriented simulation software package based on the SGI Performer 1.2 application development environment.

WEAPON SYSTEM SENSOR INTEGRATION
FOR A DIS-COMPATIBLE
VIRTUAL COCKPIT

I. Problem Statement

1.1 Introduction

The AFIT virtual cockpit (VC) is a research initiative sponsored by the Advanced Research Project Agency (ARPA). The research is investigating the feasibility of low cost, large force, distributed network simulations. My thesis effort expanded the capabilities of the prototype VC version 1.0 (VC 1.0) by developing both radar and forward looking infra-red (FLIR) sensor capabilities for integration with the VC's weapon systems. Refinements to existing software also allowed the VC to comply with the world reference model of the Distributed Interactive Simulation (DIS) version 2.0.3 standard.

The VC applies current graphics technology to a distributed interactive simulation and creates a flight simulator in a shared virtual environment. These techniques include detailed model descriptions, accurate control response, rapid frame rates, and an immersive presentation system. Successful application of these virtual environment techniques allow the operator of the VC to react and make decisions within flight scenarios that are similar to the reaction and decision making processes in the actual flight environment.

Students in any flying command experience a variety of virtual flying environments. Often the student is given access to high fidelity computer generated imagery or terrain model board visual presentations within a six degree of freedom full motion simulator. At the other end of the spectrum is a rather crude, but commonly used environment advocated by nearly all instructor pilots in the Air Force. I used it quite extensively with each of my students when I told them:

> Stanley/Zelda, sit in your kitchen tonight, strap your checklist to your leg, grab your bathroom plunger, and chair-fly the flight profile you have planned for tomorrow.

Even though the student's kitchen is lacking a realistic visual presentation, a working aerodynamic model, and an accurate cockpit environment, the instructor knows mental familiarization and review of the expected scenario can aid in successfully accomplishing the goals of the planned mission.

The goals of the VC research were to explore the requirements and potential problems in implementing the evolving Department of Defense Distributed Interactive Simulation (DIS) standard. Low cost computer graphics workstations driving virtual reality cockpits could allow large numbers of operators to simultaneously interact in a common scenario to examine application of tactics, explore new battle concepts, and give battle staff planners insight into the expected result of a planned action. The closer the virtual environment could meet the actual reality, the closer the modelled action could reflect an actual event.

The previous VC 1.0 displayed aircraft control and performance instrumentation, as well as a simple head-up display (HUD). While in this development stage, the cockpit allowed an operator to act only as an airborne observer. The planned actions of a DIS user were not only to be an observer, but to act as a participant in the realistic, large force scenarios envisioned by ARPA. The VC 1.0 required the addition of a weapon system before it could participate in a cognitive, instructional or evaluative role. This thesis effort remained within the ARPA guidelines for a low cost, realistic flight simulator which could fill the void between current high-end and low-end simulators. The VC 2.0 can run on systems costing less than $250,000, uses off the shelf hardware and components, and operates with AFIT developed software.

The fulfillment of the goals for this research entailed the software design and development of a target-detecting air-to-ground and air-to-air radar which augments the user's visual acquisition of simulated objects. The targeting information is coordinated with the HUD and a new FLIR display which allows accurate targeting and delivery of various munitions. While implementing these changes, the VC 2.0 also transitioned from the SIMNET Cartesian world coordinate system to the World Geodetic System 1984 (WGS-84) Standard geocentric coordinate system. The VC's image rendering and frame rate display speeds were monitored to assess the impact of adding these increased capabilities and modifications. These program specific algorithms allowed the VC 2.0 to render consistently at nearly fifteen frames per second (fps).

1.2 Background

The VC extends synthetic environment research begun by AFIT's Electrical and Computer Engineering Department in 1988. The first research efforts investigated low cost virtual environment methods. These efforts resulted in the ability to view simulated computer generated virtual environments with a helmet-mounted-display (HMD). The HMD was constructed from off the shelf hardware, with the exception of a Polhemus head tracking device, and cost $1000. (30:80). By the end of 1990 the department investigated the effects of hosting a flight simulator on inexpensive computer image generators (CIGs) while interfacing through an HMD (26:940). The next major development, completed in 1991, was the creation of a graphics database management system supporting synthetic environment research (5). A specific application for use with the graphics database was an expandable, object-oriented, testbed flight simulator (35). ARPA's interest in low cost distributed interactive simulations prompted the completion of the VC 1.0 which was built on the framework of the previous object-oriented flight simulator. This current simulator successfully participated in ARPA-sponsored WARBREAKER exercises, in particular the Zealous Pursuit demonstration program (24:2-3,5-1)(33:4,44).

The emphasis on low cost methods is a result of a recognized need for large force command and control exercises. Simultaneously, the Department of Defense (DOD) is experiencing budget restrictions. The reduction in funds to regularly employ actual forces results in the requirement for attaining learning objectives with low cost training methods. The current technology flight simulators cost less than the actual aircraft they represent, but are still costly when purchased in large quantities. Barnes states that current simulators are costly because they are built to attain high levels of realism and accuracy across a broad range of training objectives. These training objectives involve both physical and mental processes (2:65).

Barnes states that a flight simulator can be adjusted to attain the particular objective the instructor is seeking. Virtual environment technology is an inexpensive method of creating realistic scenarios for developing coordination, increasing perception, and gauging reaction. The virtual environment also allows real-time interaction with immediate feedback developing the user's cerebral skills of understanding, learning, investigating, and optimizing. Experienced users do not require a one-to-one accurate description of the tool they will employ, but need a realistic presentation of the scenario in which they are immersed (2). The virtual cockpit is an ongoing development platform to create the necessary tool the operator can employ in large force exercises. ARPA's long range goal is to provide numerous simulation platforms to interact in a realistic scenario so experienced operators can economically participate in large force exercises.

1.3 Problem Components

The design goal for this thesis effort was to provide symbolic radar and FLIR presentations at an acceptable data processing and image generation cost.

1.3.1 Realistic Frame Rates.

The CIG must create a realistic virtual environment when presenting the scenario. Previous research showed the minimum acceptable image frame rate is possible with a moderate cost commercial graphics workstation like the Silicon Graphics Iris 4D, but not with a high-end personal computer augmented by specialized graphics processors (26:944). The minimum target frame rate established by the AFIT graphics research is 10-15 frames per second. When below this frame rate threshold, the computer generated image begins to lose its appearance of smooth motion (13:180).

The appearance of smooth motion in the flight simulator is critical to the operator's perception of a realistic environment. High performance flight simulators typically display thirty frames per second when presenting critical phases of flight (13:919). AFIT initially conducted frame rate research using a commercial game production flight simulator which contained a limited description of the maneuvering environment. The environment consisted of large, low detailed polygons representing the ground and accompanying structures.

1.3.2 Realistic Scenarios.

The VC maneuvers in a realistic environment built from a real world database. Because VC 1.0 could act only as an airborne observer, there were few benefits to using it in distributed network simulations. Giving the user an ability to actively participate in an actual mission role was the objective behind creating a realistically operating platform. A recognized component for the simulator in proposed combat simulations is a weapon system capability. Before successful employment of the weapon system, the user must have a method of identifying valid targets. The VC 1.0 method for target location required the operator to visually acquire targets while looking through the VC's wind-screen. Without a sensor capability, the operator is strictly limited to close range, visual engagements. This target acquisition limitation does not contribute to the accuracy of the scenario, nor does it provide realistic training.

1.3.3 Cost of Expanding Capabilities.

Moderate cost CIGs proved acceptable in early evaluations, but these CIGs are still subject to degradations in image frame-rate performance. Image rendering rates are affected by the number of polygons used in the modelling description and by the level of detail displayed for a particular object. Detail is achieved by either applying a texture to a polygon description, increasing the amount of polygons used in describing the object model, or a combination of both techniques (13:741). The simulator in its VC 1.0 configuration ran at an average of five to six frames per second. This rate was below the target threshold. Any new components added to the cockpit could increase the level of detail, add new processing requirements, and would therefore slow down the image rendering speed. Other thesis students were simultaneously investigating methods to increase the CIG frame rate while this effort added new capabilities. Any additional capabilities added with this effort could not create an unsolvable problem in the search for increased frame rates.

1.4 Approach/Methodology

The original object-oriented AFIT flight simulator was designed under the assumption the simulator would act as an expanding testbed for developing technology. The simulator was created with the expressed intention of increasing its capabilities through constant modifications. Ease of reuse and modification was the driving force behind designing the simulator using an object-oriented methodology. I would continue this object-oriented, incremental build approach in the modifications I designed.

1.4.1 The Radar Display.

A suitable display area and radar screen representation were the first design goals. A simple radar representation would be the first objective, and it was modelled upon current fighter aircraft systems (Figure 1) (11:1-99).

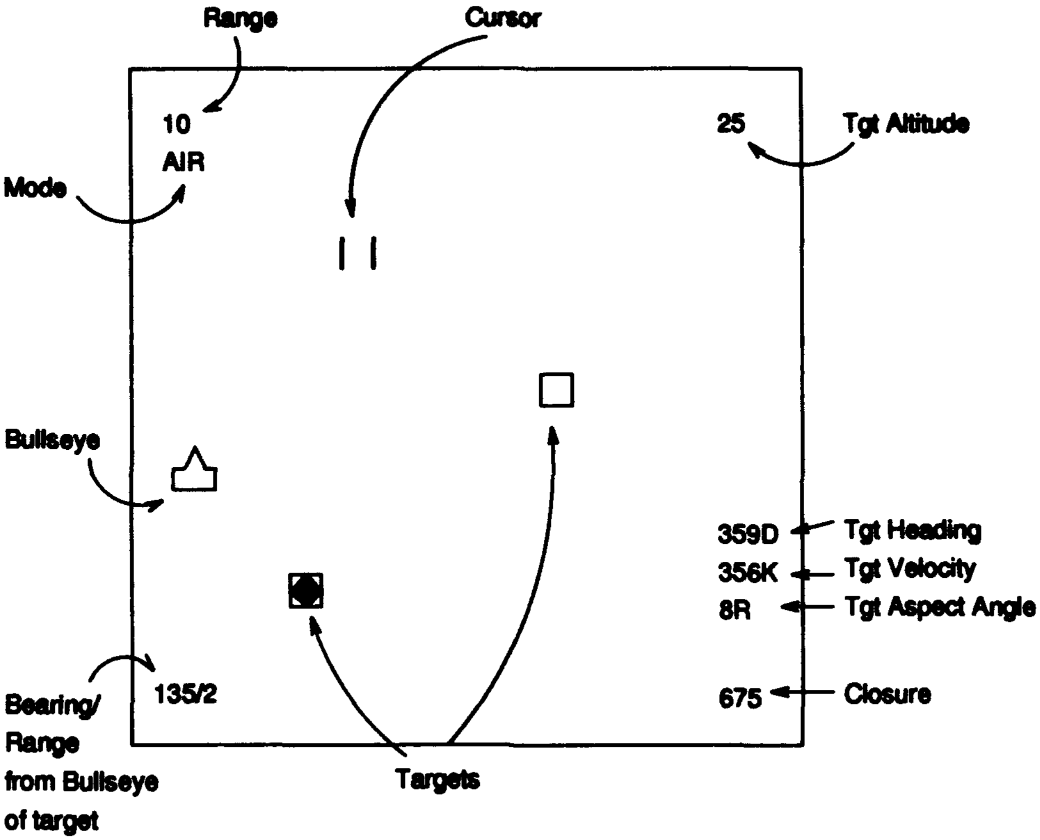


Figure 1. Simplified F-16 Air-To-Air Radar Display

The radar display projects a fixed view along the aircraft longitudinal axis corresponding to the field of view observed from looking forward through the front wind-screen. The display is roll-stabilized so any inputs around the longitudinal axis will not affect the presentation. Enhancements to the display presentation include the ability to change the radar ranging and discriminate between air and ground targets in air-to-air or air-to-ground modes.

1.4.2 Target Identification. Within the radar display area, target objects are placed in their appropriate locations. This corresponds to the acquisition of a radar return. The display allows the user to choose different potential bogies (unidentified radar tracks) as the target of interest and therefore lock-on and track the chosen target. Different icons representing the designated target versus other radar returns completes the target identification feature which allows quick recognition of the moving target among other returns (Figure 2).

Converting the three dimensional description of targets into a two dimensional view projection is the difficult task. A simplified over-head view of the aircraft flight path is the initial radar presentation. Eventually we examined other computationally costly imaging techniques to render more realistic presentations and enhance the basic display.

The simulator and all objects in the radar field of view can have their center of mass coordinates translated to the VC reference frame with one transformation. This transformation is accomplished by subtracting the VC's x, y, z position vector from all objects' position vectors. This translation aligns all target objects in the same frame of reference as the VC. The VC center of mass now acts as the origin

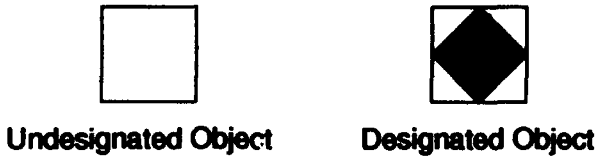


Figure 2. Radar Object Symbols

and corresponds to the bottom center of the radar display. Range and azimuth vectors are then available to the target objects at their new translated coordinates. These vectors are scaled to the appropriate distance so an object representation will be placed on the corresponding coordinates of the two dimensional radar display surface.

This simplified rendering method does not allow the depiction of background radar clutter which is generated from the natural environment. Imaging methods such as ray casting or simultaneous image projection might allow a more detailed display. The problem with these more detailed imaging methods is the increased burden to the CIG's CPU processors. Detailed imaging increases the processing requirements and results in slower frame rates since the graphic rendering routines are waiting for the completion of the CPU computations. A tradeoff in decreased radar realism may be allowed if the complexity is too great when implementing more realistic presentations.

1.4.3 FLIR Requirements. The FLIR allows the user to visually identify and target particular objects or positions in the environment. This requires rendering an alternative view of the existing scene from a viewpoint other than the pilot's cockpit position. The image rendering techniques can also create a view with color and shading characteristics different than images based on normal visible light. Modifications to the view through magnification and view frustum manipulation allow increased resolution and target/navigation coordinate updates.

1.4.4 Weapon System Interface. A weapon system interface allows the user to select a particular target from the radar display. This target selection provides azimuth steering to a weapon delivery system. When a target is selected, the appropriate target coordinates are passed to the delivery system so the delivery system can generate accurate weapon ballistics. A well-defined interface specifying parameters used by the HUD object for aiming, and parameters used by the weapon system for ballistic computations allowed concurrent development of separate thesis efforts.

1.4.5 WGS-84 Changes. The VC 1.0 operates using the SIMNET protocol data unit (PDU) standard. The migration to compatibility to the DIS standard requires not only a more detailed PDU, but a different position and orientation representation.

The SIMNET position description is from a "flat" earth surface description of a right-hand Cartesian coordinate system with the surface origin at the lower left-hand corner of the terrain rectangle. All vehicle orientations are described by a 3X3 rotation matrix specifying the relative rotation between the vehicle's coordinate system and the world coordinate system (27:21).

The DIS position description is also defined in relation to a right hand Cartesian coordinate system, but the modelled terrain is an actual round earth description specified in the WGS-84 standard. All positions and modelling have to correspond with a "round" earth description and orientations as described in the standard aerodynamic definition of the three Euler angles *psi*, *theta*, and *phi* (18).

1.5 Conclusions

The design goal for this thesis effort was to provide symbolic radar and FLIR presentations at an acceptable data processing and image generation cost. These presentations work while meeting the requirements stated in the DIS standard. This was accomplished by defining and rendering the minimum presentation symbology required by a simulation participant operating radar and FLIR displays. Meeting the minimum requirements created a VC able to interact in ARPA's realistic large force distributed simulation scenarios while being operated by experienced weapon system users.

II. Literature Review

2.1 Introduction

The United States Air Force is challenged by decreasing budgets to do more with less. The Air Force mission is essential to the defense of the nation and cannot be allowed to atrophy, which means creative methods to maintain force readiness are required. Research directed by the Department of Defense is continually looking for cost effective methods to conduct the mission. A method of training and an area of research pioneered during World War II is instruction with simulated aircraft. Simulation is a means to provide effective, low cost training. Even greater savings can be realized by the judicious use of a variety of virtual environment systems for aircraft simulation. A quick review of aircraft simulation history will show the current state of simulation technology and provide a baseline for comparison of the new technologies. New ideas concerning necessary simulation requirements will then show how different virtual environments can meet specified training objectives.

2.2 Flight Simulation

Flight simulation has enjoyed popularity with the flying services since the introduction of Link training aids. Over the last fifty years great advances and widespread uses of flight simulation have developed. Simulators offer advantages over actual flight operations in areas of cost, safety, opportunity, and ecology (32:234-236).

The unit cost of an aircraft has increased dramatically since the Wright brothers built their first production models. Because the average cost of a B1-B exceeds $205 million, it becomes increasingly prohibitive to fly the aircraft solely to fulfill training requirements (25:C-1). Any excess use over mission requirements is an added burden on operation and maintenance costs. Rolfe cites research by Orlansky finding simulators cost 30-65% of their aircraft counterparts, require only 8% of an actual flying budget, and recoup their purchase price after about two years of use (32:234).

Simulators also protect aircraft from catastrophic loss while performing hazardous maneuvers. Before the adoption of simulators for effective emergency procedure training, more accidents occurred while practicing an emergency procedure than resulted from actual occurrences of the emergencies (32:235).

Effective airborne training requires coordinating a variety of events. Examples of these requirements include scheduling a suitable airspace or range, maintenance units providing a ready airframe, and the weather allowing the plan to proceed. Often a training exercise is scheduled months in advance without accurate knowledge of potential conflicting events. While training toward a particular enemy objective, any pilot would relish the opportunity to practice a maneuver in a realistic combat scenario before actually employing that maneuver in a hostile zone. Simulation offers an environment that avoids the limitations on training flights that result from unforeseen scheduling restrictions or hostile forces.

Recent decades have witnessed dramatic increases in fuel prices combined with a heightened interest in environmental concerns. Operation costs rise with increasing fuel prices while restrictions are placed on access to training areas. Restrictions result from people's heightened sensitivities to noise, worries about effects on the environment, and perceptions of danger to residents living beneath flying areas. The use of flight simulators simultaneously reduces fuel consumption while limiting disturbances to the local populace. While the simulator is relied upon to relieve the increased pressures on the training environment, a simulator is useful only if the training objective can realistically be attained through the simulator medium.

2.3 Virtual Environments

A single definition of virtual environments is difficult to find. Many descriptions of virtual environments and virtual reality exist. Coalescing many of the common terms used in these descriptions can give an idea of the essential characteristics of a virtual environment.

According to Rheingold, a virtual environment is a setting in which a person is "surrounded by a three-dimensional computer generated representation, and is able to move around in the virtual world and see it from different angles, to reach into it, grab it, and reshape it" (31). Other commentators add real-time, view-centered, head-tracking perspective with large angle of view, interactive control, and binocular display to the description (8:135).

When the term virtual reality is expanded to virtual environments, new criteria are added including touch, smell and sound (3). Depending on the chosen definition, any system could entail a majority of the descriptive requirements, but not all of them. No matter what the definition, specific devices and effects are required to create immersive virtual environments.

2.3.1 Projection Mechanisms. Virtual environments present the viewer with simulated images which replace the physical objects used while performing an action. Research into virtual environments was sparked when Ivan Sutherland proposed his *Ultimate Display* in 1965 (41:506-508). He visualized computer images that could completely envelope a user's senses. Various technologies are used when presenting a virtual environment. The most common technologies used are cathode ray tubes, head-mounted displays, binocular omni-oriented monitors (BOOMs) and domes (9:66-67). Each of these technologies compares differently when they are judged by how effectively they immerse a viewer in the simulation environment, and by how clearly they present an image.

2.3.2 Immersion Effectiveness. Effective simulators using virtual environment technology rely on two characteristics of the simulation: suspension of disbelief and viewer-centered perspective (9:65). Suspension of disbelief is the ability of the simulator to cause viewers to "give in" to the simulation and believe they are actually there while they are concentrating on the application. Consequently immersion results in the viewer ignoring the simulator's projection medium. A viewer-centered perspective simulates the view from where the viewer would be located in the actual environment. Creating an image that is not centered on the viewer makes suspension of disbelief increasingly difficult. When viewers can finally discount the viewing medium used by the simulator, they are then immersed in the simulation. "Immersion is the degree of visual simulation a virtual reality interface provides for the viewer—the degree of the suspension of disbelief" (9:67).

The five most critical factors when determining immersion's effectiveness are field of view, panorama, viewer-centered perspective, body and physical representations, and intrusion (9:67-68). Immersion comparison for the different technologies is listed in Table 1.

Field of view represents the visual angle viewers can see without rotating their heads. Panorama is the ability of a display to surround the viewer. Panorama differs from field of view in that panorama takes into account the rotation of the viewer's head. View-centered perspective is determined from sensing the location of the viewer. Quick detection of movement and accurate measurements are desired. Body and physical representations account for how much of the viewer and corresponding viewer-handled objects are physically seen, or will require a simulated presentation. Any object that should be seen but is not physically within the viewer's line of sight requires the generation of a simulated image. Intrusion corresponds to the restrictions the viewing device places on the viewer when using the equipment.

2.3.3 Visualization Effectiveness. A system is judged not only on how well it immerses a viewer, but also by how clearly and realistically the system projects the image. These criteria determine the second area of comparison termed visualization. Many factors contribute to visualization but two are of prime importance to flight simulators: visual acuity and look around (9:68). A visualization comparison for different technologies is listed in Table 2.

Visual acuity rates a display system in methods similar to how an optometrist determines a person's vision when giving an eye test. The display system's ability to generate fine levels of detail is evaluated by comparing the width of projected pixels and the distance the projected pixels are presented from the viewer's eyes. Look around describes the system's ability to project an image which shows movement about an object allowing views of the object from different look angles (9:69).

2.4 Current Technology

Domed simulators are the preferred advanced training aids used by tactical flying units. A review of a dome's critical comparison factors shows why domes are the dominant virtual environment technology in use today. Examining immersion criteria for domed simulators shows that a dome does not limit a viewers' natural field of view, and because a dome continually projects 360 degrees around the viewer, there is no delay in projecting panorama. When viewers decide to look over their shoulders, they see a projection that is continuously computed and displayed. A dome usually provides a mockup of the aircraft cockpit which eliminates any near perspective problems. The body representation corresponds to the viewers' true physical body. As a result of this representation, no conflicts exist with projected images because the near surroundings of the cockpit are also physically implemented. Finally, the dome does not place any physical limitations on viewers except for those encountered in an actual aircraft cockpit.

The visualization criteria show that the look around for domed displays is not a factor because any close objects are physically present in the simulator cockpit. The dome's lack of visual acuity is offset by its greater immersion capabilities. A dome can also project in higher detail a local area of interest while leaving peripheral images at a lower acuity resolution (28:186) (32:155).

<table>
<tr><th colspan="6">Immersion Issues</th></tr>
<tr><th></th><th>Field of View</th><th>Panorama</th><th>Perspective</th><th>Body Rep.</th><th>Intrusion</th></tr>
<tr><td>CRT</td><td>45°</td><td>None</td><td>Slow</td><td>Physical</td><td>None</td></tr>
<tr><td>BOOM</td><td>90° ⟷ 120°</td><td>Fast</td><td>Fast</td><td>Virtual</td><td>Partial</td></tr>
<tr><td>HMD</td><td>100° ⟷ 140°</td><td>Slow</td><td>Slow</td><td>Virtual</td><td>Full</td></tr>
<tr><td>DOME</td><td>Full</td><td>Fast</td><td>Fast</td><td>Physical</td><td>None</td></tr>
<tr><td colspan="6">(9:69)</td></tr>
</table>

Table 1. Comparison of Immersion Issues.

<table>
<tr><th colspan="3">Visualization Issues</th></tr>
<tr><th></th><th>Visual Acuity</th><th>Look Around</th></tr>
<tr><td>CRT</td><td>20/40</td><td>Limited</td></tr>
<tr><td>BOOM</td><td>20/85*</td><td>Full</td></tr>
<tr><td>HMD</td><td>20/425</td><td>Full</td></tr>
<tr><td>DOME</td><td>20/110</td><td>N/A</td></tr>
<tr><td colspan="3">* 90 degree field of view, black and white, source (9:69)</td></tr>
</table>

Table 2. Comparison of Visualization Issues.

2.5 New Trends

Domes appear superior in all categories for simulating flight, but require substantial investment compared to other simulation methods due to the nature of a dome's complexity. When comparing cost and space to other methods of creating virtual flight environments, a user may desire to make tradeoffs on immersion and visualization issues. Domes usually require multiple image channels and complex projection equipment. They are built with exacting detail replicating the aircraft cockpit and weapon systems, and in some cases provide a full motion platform. Domed simulators are often located in buildings totally dedicated to housing the simulation machinery, which can result in the total cost of a single simulator exceeding $20 million (32:154).

Implementing alternative methods of virtual flight environments has reduced simulation costs by two orders of magnitude. The reduction stems from using a single, low to moderate cost computer graphics workstation which reduces the machinery cost and the physical area required for the simulator. The required space is often only $1/100^{th}$ the area of a dome complex (42:2). The primary drawback of the alternative methods is the limitation on the accuracy in replicating the simulated environment. In some training situations, a realistic scenario is of primary importance and the viewer does not require a highly accurate display. When given the opportunity to emphasize realism, a tradeoff toward lower accuracy and cost will not adversely affect the realistic training objective (17:2). Many training objectives and learning processes lend themselves to an environment that does not require exacting accuracy when replicating the aircraft environment. These objectives and processes are prime candidates for the less restrictive, cost beneficial virtual environments (2:65-71).

2.6 Network Interaction

Advances in technology have allowed the interaction between numerous simulation entities within the same environment. This allows physically separate simulators to operate as if they were next to each other and interact real-time in selected scenarios. The Army spear-headed this effort with the SIMNET protocol for large scale combat training exercises beginning in 1983.

SIMNET provides for fully-manned platoon-, company-, and battalion-level units to fight force-on-force engagements against an opposing unit of similar composition. Furthermore, it does so in the context of a joint, combined arms environment with the complete range of command and control and combat service support elements essential to actual military operations. This allows combat units to benefit from collective, combined arms, joint war fighting skills at a fraction of the cost of an equivalent exercise in the field (27:1). Three research areas SIMNET addresses include:

1. Better and cheaper collective training for combined arms, joint war fighting skills.
2. A test bed for doctrine and tactics development and assessment in a full combined arms joint setting.
3. A "simulate before you build" development model.

The Navy has investigated distributed network technology at the Naval Postgraduate School (NPS) with their NPSNET environment. NPSNET allowed the AFIT VC to interact with NPS developed network entities. The culmination of this effort was a week long demonstration of distributed interactive simulation at the SIGGRAPH 93 conference in Anaheim California. At this conference the NPS and AFIT connected over a distributed network using a modified Distributed Interactive Simulation (DIS) network protocol that evolved from past research on the SIMNET protocol. Full compliance with DIS required a world reference frame modification to implement a round earth coordinate system.

In 1989 a research thrust began developing a more robust simulation standard that is known as DIS. DIS incorporates all the essential elements of SIMNET with added emphasis on increased ranges of heterogeneous simulators, wider application of software reuse, and an open architecture easing future expansion. An important aspect of the DIS development is the effort to recognize DIS as an international standard for distributed simulation. The current DIS v2.0.3 is being submitted for review (40:10). Departures from SIMNET include a coordinate system based on the World Geodetic Survey 1984, an entity coordinate system based on an aerodynamic description, and an entity orientation using the aerodynamic Euler angles *psi*, *theta*, and *phi* (18). The current goals for DIS include supporting thousands of interactive real-time entities broadcasting across the same network using the protocol. The Zen Regard exercises of the ARPA Warbreaker program are attempting an 8000 entity demonstration in late 1993.

2.7 Conclusion

The Air Force will continue its use of simulation when training towards and maintaining high force readiness. The Air Force can categorize training objectives based on the desired level of accuracy and realism required by the simulation. Because low cost virtual environment technologies can provide the necessary realism desired for the training objectives, the Air Force will benefit from the cost advantages gained by alternative virtual flight technologies.

III. Design Decisions

3.1 Introduction

Defining the goals of the Virtual Cockpit (VC) refined the requirements for an effective interaction by an air operator within a distributed simulation. The VC will need a minimum weapon application set and the ability to both employ the weapons and give the operator the information to make valid decisions. All operational USAF aircraft give the operator these capabilities specific to their particular mission. If the operator using the VC can interact in real-time with these basic capabilities, then the real-time decisions and applications by individuals and coordinating units can be analyzed. The limiting factor for the VC is that it is not a training aid in helping the operator learn *how* to use a specific weapon or system, but it is an aid to better understand the intricate interactions in large scale planning and employment exercises. The decisions for the VC weapon system applications had to be balanced between the desire to effectively model the operating environment while facing the restricted computational power of the workstation processors.

I will describe how an understanding of the operating environment resulted in refining the need for the development of several C++ classes that model actual aircraft objects. The objects include a multi-function display (MFD) which evolved from the initial desire for a radar display, an IR display, and a navigation panel. Another object is an interactive cursor which creates the interface between the operator and the MFD. The final major object is an inertial navigation system (INS) which is the computational engine that solves the calculations required to drive the various displays. The INS also provides state inputs to the navigation display and weapon systems (figure 3). Operational testing of the VC and planning for future applications resulted in the definition of additional requirements. These included a moving parts object, an autonomous forces object (figure 4), and a utilities class that handles an AFIT developed solution to the round earth DIS implementation.

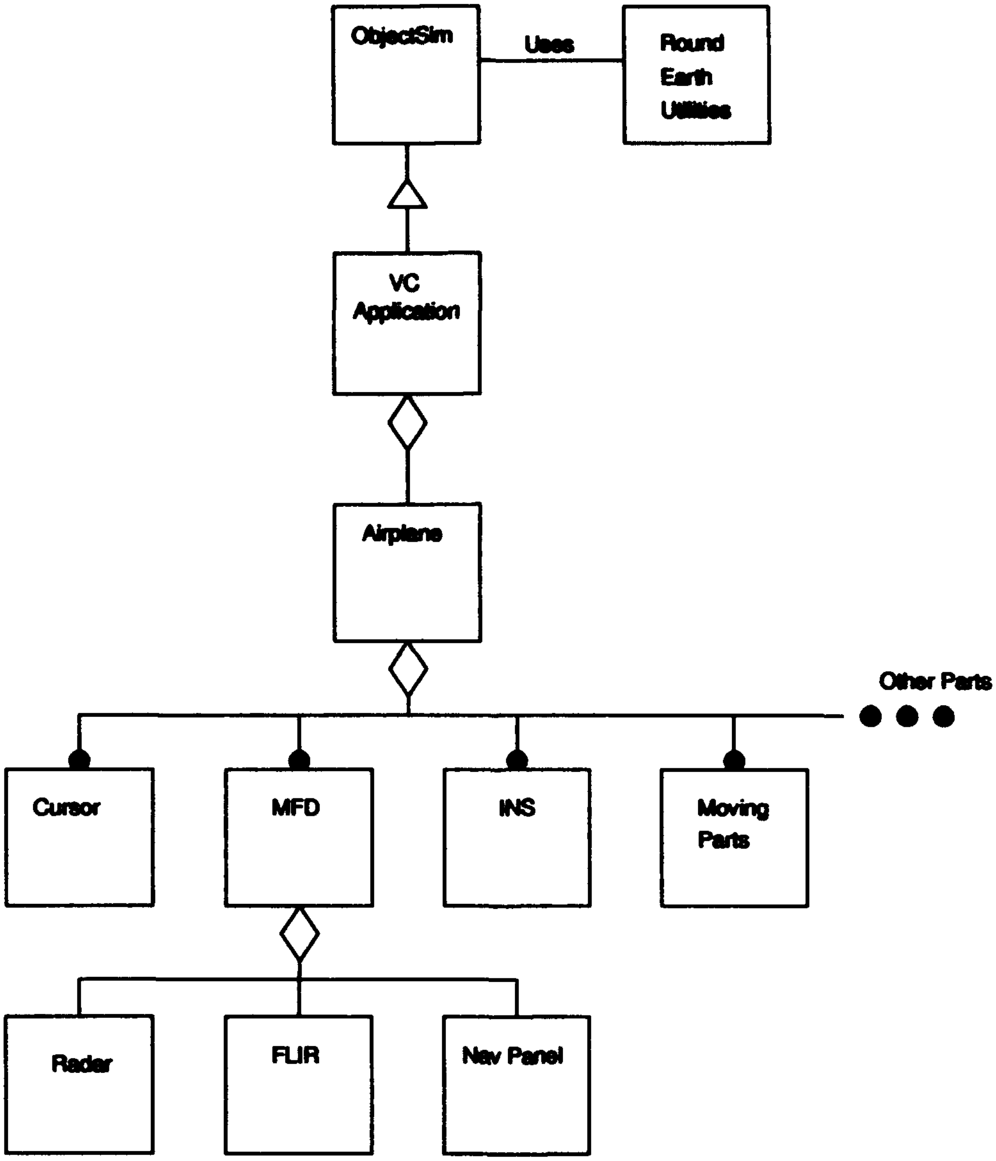


Figure 3. Virtual Cockpit Object Diagram Of Thesis Software

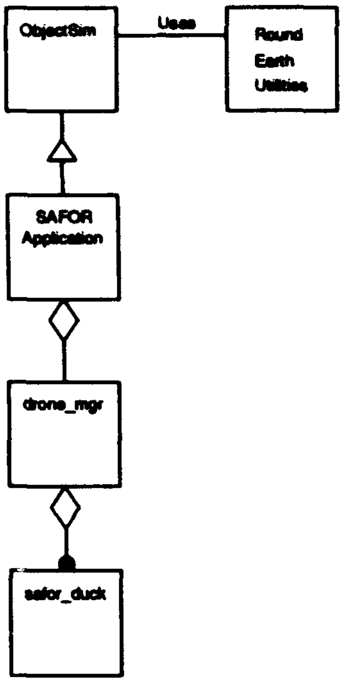


Figure 4. SAFOR Object Diagram

3.2 Analysis

Radar implementation must consider the computational power of the available machines. Actual aircraft subsystems on modern fighters consist of several embedded systems that are specifically programmed and tuned to support the operational requirements expected for the mission. A typical set of aircraft subsystems looks like figure 5 (1). This figure resembles the various weapon systems and components needed for a minimum subset of VC operations. The disadvantage for the VC when compared to the specifically designed algorithms and hardware of current avionics systems is the limited computing speeds of the SGI workstations. The SGI workstation is not totally dedicated to solving one particular avionics algorithm bounded by a hard real-time limit. For example, the SGI by itself must manage all network interfaces, operating system functions, computationally intensive application programs, and the graphics rendering.

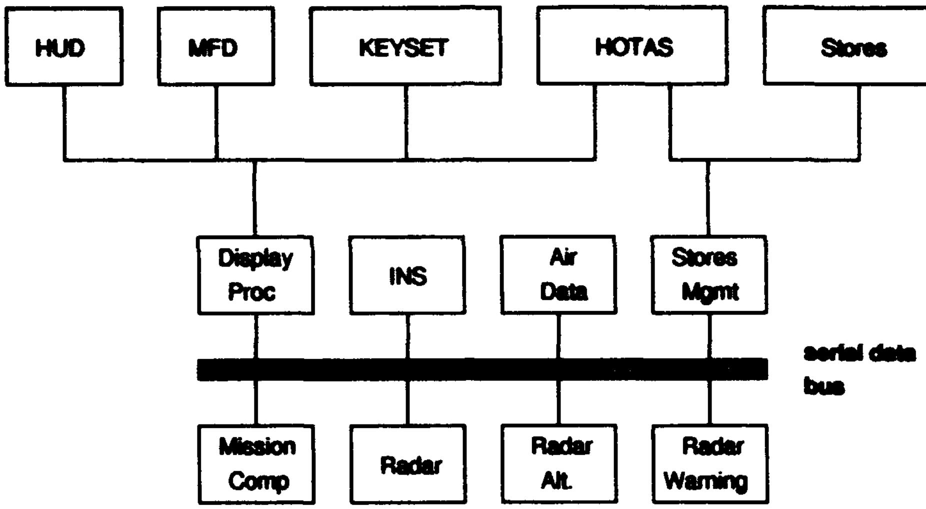


Figure 5. Example Avionics System Structure (1)

The emphasis on the development of an air-to-air radar originated from three main factors. Two factors emphasize the difficulties inherent in air-to-air engagements, and one emphasizes the relative ease in which air-to-ground operations are handled.

1. The hardest simulation entities to locate and interact with were other air entities because of the large area of the air environment, the small size of the air target, and the speed at which the air targets moved.
2. Weapons engagements for air targets requires target range, location, and state information be displayed for accurate user decisions. Target state variables could include energy, intentions, and degree of threat.
3. Ground targets are either stationary or move at a relatively slow rate. Therefore ground target movements are fairly predictable and the location of the target is aided by landmarks visible on the ground.

When defining the minimum operator requirements for the VC, analysis of and experience with actual weapon systems was done. I drew from my own experience as primarily an air to ground operator with 720 hours in the F-111F and 2700 total flying hours. For air-to-air input and as a supplement to my air-to-ground experience, I conducted interviews with other AFIT students with experience in the F-16, F-15, and F-4G. I also conducted interviews with the 89^{th} FS, an Air National Guard unit at WPAFB that operates the F-16 and which is converting from an air-to-air role to a dual role mission (air-to-air/air-to-ground). Face to face and telephone interviews were conducted with operational flyers of the F-15E from the 334^{th} FS of Seymour Johnson AFB.

All flying missions require a minimum information set that allows an operator to make decisions dictated by the local operating environment. This information is used to build what is termed in the flying world as situational awareness (SA) (39).

3.2.1 Situational Awareness. SA consists of everything the operator knows of the environment; weather, terrain features, own-ship location, location of friendlies, location of bogies (unknown identity), location of bandits (the enemy), employment tactics, friendly/threat performance and capabilities, and anything else that is known at a particular instant. Knowing what all the factors are determines the operator's SA. High SA (lots of facts and cognizant of the relationship between the facts) increases the ability of the operator to make a "good" decision. A good decision usually results in the effective employment of the system. Low SA (limited facts or poor understanding of the relationship between the facts) limits the operator's decision-making ability and seldom results in the optimal employment. Operator experience can enable better decisions despite limited information, but experience can seldom completely overcome a lack of critical information.

3.2.1.1 Own-ship Aircraft State. One set of critical information is gained from understanding the own-ship position and state. Standard control and performance indicators give a quick indication of the aircraft state. Knowing own-ship position requires knowing the location of a specific reference point and some means of determining displacement from that reference. Flight planning requires the selection, computation, and review of a selected flight sequence (6:3-4). The flight sequence is planned using recognizable navigation points. The sequence of navigation points is then used to guide the operator in the mission. Modern aircraft are programmed with route points and other points of interest (bulls-eyes) in the navigation computer or inertial navigation system (INS). The positional awareness requirement emphasized the need for a rudimentary navigation display for selecting and showing points of interest. A means of calculating time, distance, or heading to the point of interest within the INS is also needed.

3.2.1.2 "Bogey Dope". The operator is interested in the location (absolute or relative) of an object of interest. State variables such as speed, heading, altitude, and degree of threat are also nice to know. This information is available from friendly systems such as an Airborne Warning and Control System (AWACS), ground control intercept (GCI) operator, or on-board detection systems. Analyzing the radar returns allows the radar weapon system to compute specific parameters. The parameter values and associated visual cues are then projected on the radar display or other instruments to aid decision-making. With these indications, the radar operator gains insight about the threat's actions, intentions, and capabilities.

The insight provided by the radar system is then used to take appropriate offensive or defensive actions. The operator can employ a weapon against the target by coordinating the detected information with the specific weapon system selected. A defensive action could include the quickest way to negate the potential threat by maneuvering in an extension (running away bravely), or using known environmental factors or knowledge of threat weapon systems to evade detection (terrain masking or stopping closure to negate adversary radar systems).

3.2.1.3 Target Information. The information displayed about a target can be different according to the specific target type selected (air or ground). An air target's location is routinely specified as a relative position from a known location. This relative position is normally formatted as a compass heading from

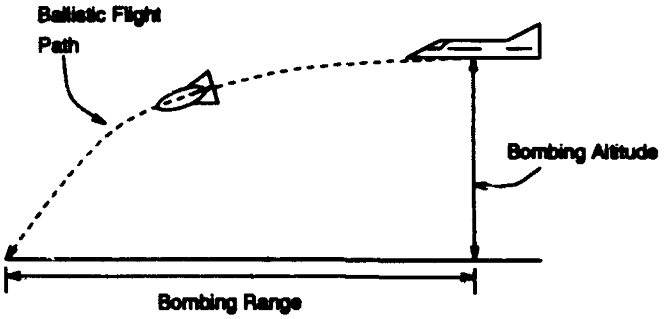


Figure 6. Ballistic Bomb Fall Diagram

the known location for a specific distance. If all friendly participants are using the same common reference location, known as a bullseye, then a radio transmission from a radar operator can succinctly impart the needed information. An example of an AWACS radio transmission is:

`Bogey, Cherry, 250 at 25, angels 180,`

which relates the position of the bogey at 250 degrees for 25 miles at an altitude of 18000 feet from the bullseye called Cherry. Other information transmitted could also relate the speed and target type of the bogey either in code or in the clear if these factors were known.

Some target information requires the visual identification or visual application of the weapon system before employment. In air-to-air engagements this requirement usually relates to flying within visual identification range. In air-to-ground engagements this requirement may include rolling in on the target to drop a weapon visually (figure 6). Visual release of a weapon occurs when a calibrated sight shows the desired weapon impact point under the cross-hairs of a sight's aiming reticle. At the instant the sight superimposes the target, the bomb's ballistic trajectory correlates to a specific bomb range and altitude. The bomb range and altitude matches the current aircraft position from the target. A pilot should know from studying his weapon tables that a 500 pound bomb travels 3200 feet when dropped from straight and level flight at 500 feet above ground level (AGL) and 500 knots. The pilot's goal is then to be on airspeed and on altitude just prior to bomb release which *should* occur when he mashes the pickle button 3200 feet away from the target in straight and level, unaccelerated flight. In the case of EO weapons, the munition is guided to the target by directly flying the bomb body to impact with radio controlled inputs. With laser guided weapons the point of impact is continuously designated with a laser aiming device. This device illuminates the desired aim-point with laser energy which reflects back towards the bomb's sensors. The bomb then steers towards the source of the reflected energy and destroys the target.

3.3 Minimum Radar Task Set

The SA requirements outlined specifies a minimum radar task set that includes coordinating navigation information with air and ground target identification and selection.

3.3.1 Navigation. The radar screen should have preprogrammed navigation and bullseye points available for display. The reference points' relative horizontal position from the aircraft are placed in the correct position on the display. Different symbols for different types of points allows quick interpretation of the information. The information describing relative distance, heading, and closure to a point of interest should be available to allow computations for steering directions and course corrections. These indications enable accurate placement of the aircraft relative to the selected point.

3.3.2 Targeting. Awareness of all the potential targets does not allow the operator to effectively employ a weapon system unless the operator can act in a decisive manner. The operator must decide which threat is the greatest danger, select that threat from the others in the area, and employ a weapon against the threat (36).

3.3.2.1 Choosing Selected targets. The potential targets detected by the system are arrayed in their relative position from the aircraft's position. On a simple plan view radar, also described as a B-scope, the own-ship position is at the bottom center of the radar screen. Time critical target selection and deliberate target sorting and selection create the need for quick target lock-on modes and selective target lock-on modes (19). A time critical lock-on occurs when a bogey/bandit is encountered with little or no notice and the active radar must immediately acquire, lock-on, and track with little or no operator direction. Deliberate target selection can take place when an air engagement occurs with enough time to use the on-board systems to locate, sort, acquire, and track a target from a selection of potential targets.

Discriminating target selection and lock-on requires accurate cursor movement by positioning the cursor over the target of interest. Once the target is surrounded by the cursor, designating the target with the cursor will lock the radar system to the target for tracking. The radar will provide weapon system information and parameters to the aircraft displays and targeting devices. If quick reaction lock-on is required, then cursor designation without any target under the cursor will allow the radar to automatically search the illuminated airspace and locate the closest target. The radar will then lock the tracking system onto that target.

3.3.2.2 Target Information. Target coordinates and range are the necessary information for on-board weapon systems to employ ordnance against a target. The target coordinates and the aircraft's current coordinates allow ballistic computations by computers to match the selected munition's bombing altitude and range with aircraft altitude above and distance from the target. The exact instant when ballistic range matches the distance from the target is when an automatic weapons release can be signalled for accurate day or night weapons delivery. The illumination of target coordinates with laser designators allows precision guiding of munitions or precision updating of target coordinates. Identification by further sorting radar cross-section information of particular target type or size can aid a beyond visual range air engagement. Radar cross-section supplements radio interrogation methods for identification of friend or foe (IFF) reducing chances of fratricide.

Accurate employment of the weapon system against an air adversary requires state information of the adversary. Commonly used state parameters are heading, altitude, speed, closure, and aspect angle. Assessment of an adversary's state from these variables enables the operator to maneuver the own-ship against the adversary and time an attack or defense to the operator's advantage.

An adversary's altitude and speed give an indication of the potential energy of the opponent. This knowledge, coupled with the adversary's aircraft type and performance characteristics versus the own-ship's energy and performance characteristics, allows the operator to select the most advantageous maneuver at any given instant. Energy is not the sole variable in the equation. Closure gives an indication of the time available for decision making or assessment of the duration of an engagement while also giving clues to relative heading. Heading and orientation of the target combined with the own-ship heading and orientation allow accurate decisions about own-ship advantage and the degree of threat from the target.

Heading and orientation of two adversaries can be simplified into a parameter called aspect angle. Aspect angle for the VC is angle off the tail (AOT) of the bandit aircraft. AOT is the magnitude in degrees of the angle described by a pair of line segments, one which originates from the center of mass of the target and extends through the tail of the target, and the other which also originates at the center of mass of the target and extends to the center of mass of the own-ship (figure 7) (10:3-127). High aspect is deadly since the target's nose is directed toward the own-ship while low aspect is relatively benign since the target is pointing away from the own-ship.

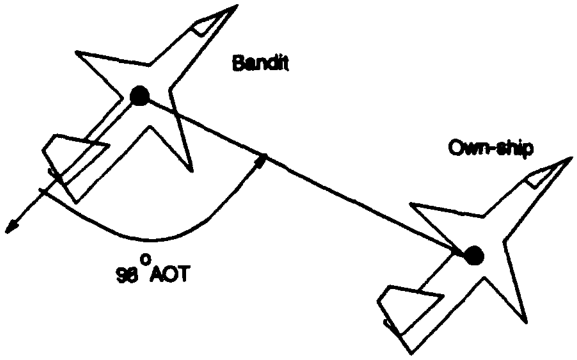


Figure 7. Aspect Angle As Angle Off The Tail (AOT)

3.4 Minimum FLIR Functions

FLIR type displays, similar to the Pave Tack systems employed by F-111F and F-4E aircraft, operate in modes that correspond to the weapon/navigation modes selected by the operator. The operator can decide which point of interest to view by selecting a target and its set of coordinates in the bomb/navigation computer systems. The FLIR will then allow the operator to manipulate the IR view and update system functions.

Operational modes of a FLIR can consist of:

- Navigation Update - Both position and velocity updates are available using precision angle and range data.
- Search/Cue - The line of sight (LOS) is controlled to locate way-points, target locations, and targets of opportunity.
- Acquisition/Recognition - A selectable display magnification allows the operator to acquire targets with normal view and then switch to magnification for enhanced target signature. This provides very effective target identification and recognition.
- Target Tracking - Precision tracking is available against both fixed and moving targets from high velocity attacks.

- Weapons Delivery/Damage Assessment - Precision angle and range data sent to the weapons computer aid in computations to provide precision delivery. The FLIR provides high quality scene imagery allowing the operator to accurately assess target damage. (14:3-6)

FLIR hardware is combined with electro-optical (EO) munitions to allow views from the FOV generated by an attached transmitter on the released EO weapon. The EO ordnance allows the operator to guide the weapon in a stand off mode by actually flying the weapon to the point of impact. This delivery method has the advantage of allowing the delivery platform to depart the target area since on board designators do not have to illuminate the desired target.

Many computational tasks were required to complete the minimum information set necessary to display the sensor situation awareness aids. Limiting the amount of computations allows the VC 2.0 to still achieve desired frame rates. The details of this process are described in the next chapter.

IV. Implementation Process

The design goals afforded many opportunities to stress SGI hardware in different configurations. The original VC 1.0 was installed on an SGI 4D 440/VGXT that ran 3 to 5 frames per second at its best with only a rudimentary HUD and without weapons systems for detection or delivery of weapons. As mentioned in Chapter III, the VC eventually had to operate simulated, embedded, real-time systems similar to figure 5. The final graphics system the VC is operating on is an SGI four processor Onyx with a Reality Engine. This state of the art graphics workstation does not have the computational ability to handle the calculations that an actual aircraft requires to operate its specialized embedded systems.

Not only do the SGI workstations lack the computing power, but the VC has to operate all detection and employment of weapon systems against objects that do not physically exist. The VC has to provide indications to the operator similar to those that are seen in an actual physical flight environment while deriving all calculations and visualization from synthetic data. The cockpit cannot physically interrogate the surrounding environment by bouncing radar transmissions off solid bodies, but must make all calculations and decisions from an existing data set available from network transmissions. The data set is derived from protocol data units (PDUs) received over a distributed network. The interface into the network is managed through the VC's encompassing ObjectSim architecture.

ObjectSim is the thesis project based on SGI's Performer graphics programming application and is summarily reviewed in Chapter 4.1.1 (37)(23). The ObjectSim network interface can currently handle 2000 network simulation entities. These entities of potential radar targets are maintained in the ObjectSim `Sim_Entity_Manager` player list. Total DIS entities are projected to number in excess of 8000 players by the end of the year. Efficient search space algorithms and methods of entity interrogation are needed to determine entities of interest so that the VC application does not become computationally bound and therefore fail to maintain realistic frame rates.

4.1 Performer Interface

Silicon Graphics' new Performer application development environment consists of subroutine code making it possible to interface efficiently into the SGI graphics hardware. With Performer the VC can take advantage of the optimized code and the multiprocessing capabilities of the workstations to increase the frame rate.

Silicon Graphics had known that most users of their hardware and software were not able to achieve the advertised performance from their products. The reasons for these inefficiencies included the user's limited knowledge of the detailed hardware architecture and the impact that poorly ordered and unnecessary graphics subroutine calls had on the graphics throughput. As a response to this difference between company specifications and actual user experience, Silicon Graphics developed the IRIS Performer application development environment. Performer combines a programming interface for creating visual simulation applications and a high-performance rendering library in a 3-D software toolkit. This environment provides an ANSI C application interface that incorporates the IRIS Graphics Library (GL) and the IRIX operating system for creating real-time visuals on Silicon Graphics systems (23:xiii).

Another AFIT thesis effort in 1993 was the engineering of a generic application that any simulation application could use to render the simulated environment. This application was titled ObjectSim (37)(figure 8). Four graphics applications encompassing six thesis efforts used the ObjectSim framework built on the Performer development environment. The applications are the Virtual Cockpit for interactive distributed simulations (12)(16), the Remote Debriefing Tool for visualizing Red Flag exercises (15), the Synthetic Battle Bridge for visualizing a battle scenario (38)(44), and the Satellite Modeller for visualizing satellite constellations (20).

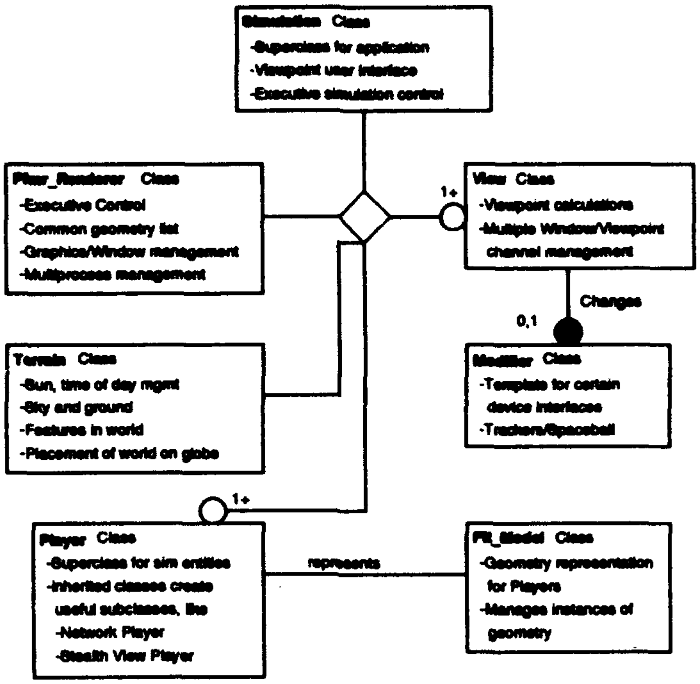

Simulation Class
-Superclass for application
-Viewpoint user interface
-Executive simulation control
Pfwr_Renderer Class
-Executive Control
-Common geometry list
-Graphics/Window management
-Multiprocess management
1+
View Class
-Viewpoint calculations
-Multiple Window/Viewpoint channel management
Changes
0,1
Modifier Class
-Template for certain device interfaces
-Trackers/Spaceball
Terrain Class
-Sun, time of day mgmt
-Sky and ground
-Features in world
-Placement of world on globe
1+
Player Class
-Superclass for sim entities
-Inherited classes create useful subclasses, like
-Network Player
-Stealth View Player
represents
Flt_Model Class
-Geometry representation for Players
-Manages instances of geometry

Figure 8. ObjectSim High Level Diagram (37:8)

4.1.1 ObjectSim Application. ObjectSim, using the Performer interface, splits the application, draw, and cull operations of the simulation across processors allowing computationally intense processes to run simultaneously with the graphics rendering. Performer coordinates process synchronization so that no single process out-paces any other graphics process. This coordination then results in smooth frame rate control. ObjectSim also provides the link between the simulation and the outside network traffic. Previous thesis efforts and current contractor staff provided AFIT with the necessary network transmit and receive processes to accept DIS network traffic and also broadcast across the network. ObjectSim accepts the network daemon information and correlates that information with a predefined model set allowing all network traffic to become visible in the simulated environment (figure 9). Each site active on the DIS network and interactive object in the predefined simulation scenario becomes a Player in the ObjectSim application. All Players can then interact with each other in the simulation.

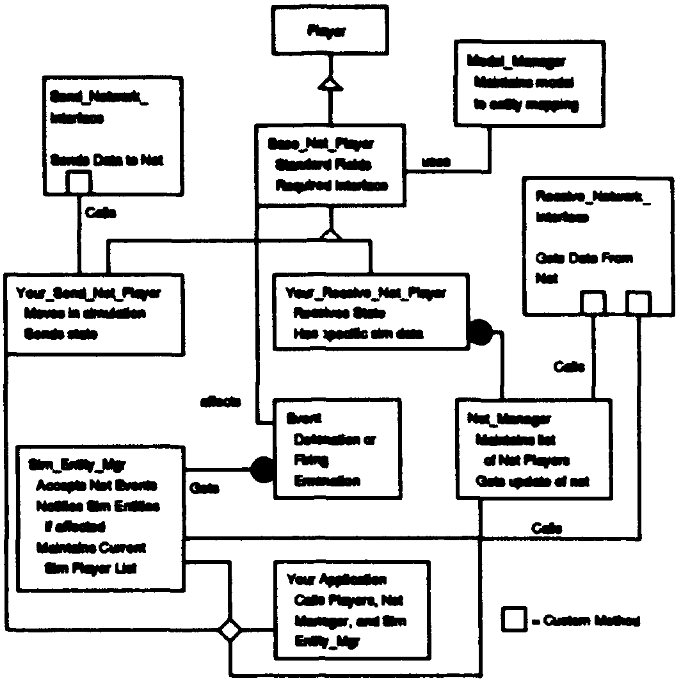


Figure 9. ObjectSim Typical Network Interface (37:38)

Visualizing the information flow then shows all the DIS Player broadcast information in the form of DIS PDUs accepted off the common network by the daemons (figure 10). The daemons process information in a shared buffer structure also accessed by the `VC_Object_Manager`. The `VC_Object_Manager` organizes the data in a standard structure to put into a shared `Player_List`. The `VC_Net_Manager` combines the network information with information about static objects in the terrain descrip-

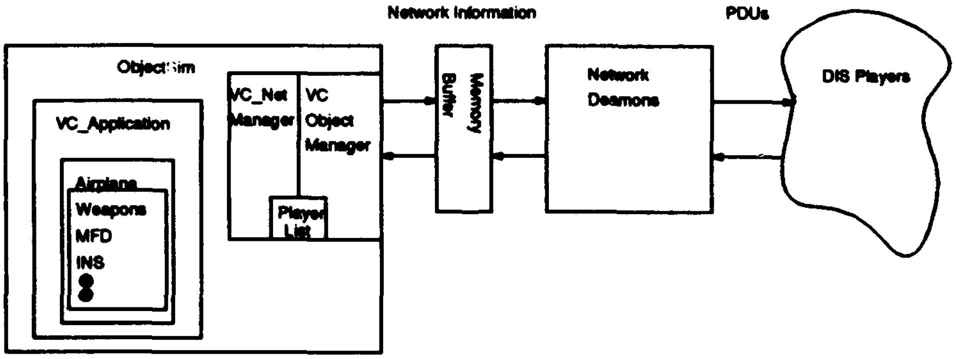


Figure 10. ObjectSim and Virtual Cockpit Application Data Flow

tion and local events and also updates the shared `Player_List`. The `Player_List` is then the sole repository of all simulation information that could affect the VC.

4.1.2 Shared Memory. The original VC did not adequately monitor and lock out simultaneous access to common data structures from separate processes since the VC used optimistic concurrency control. As a result of this control, the VC was subject to frequent system crashes. These crashes were characterized by segmentation faults and invalid values for variables in the aerodynamic model of the VC. To counter this inadequacy, the Performer shared memory initialization and locking semaphores were added to the VC to protect the common data areas. This cleared the system crashes. It also helped organize the control of data within the program. Various sections of the same program code are active in different process threads simultaneously. The multi-thread control then results in a common variable in the program losing visibility across processes. Passing the common variable through the shared memory areas ensured global visibility in the program across the separate threads of operation.

An object in the application such as the MFD could have some operations that occur only on the application thread, such as updating user inputs from the HOTAS. If the HOTAS updates are needed to determine what will be displayed on the draw thread, such as selecting a navigation radar display rather than an air radar display, the HOTAS input will not be visible from the application thread. Only when the HOTAS input is passed through shared memory can both process threads communicate through a common variable. Locks on the shared areas prevented simultaneous access to the shared areas along with standard update procedures that performed a single update routine on the shared memory at the completion of each computation cycle.

4.2 Scaled Text

A detriment to the visual realism of the simulation was the limited availability of an adequate text rendering routine. VC 1.0 attempted to provide realistic text by maintaing sets of numbers as geometric model descriptions in the simulation that were translated and rotated into the proper display position. This model manipulation was costly in computation overhead and contributed to an unacceptably slow frame rate.

4.2.1 GL Library standard.

The standard GL Library text will not rotate to the proper perspective orientations since the text always aligns to level with the screen orientation. The letters will move horizontally or vertically, but will not rotate or change in depth in relation to the screen. In addition, unless the font size is specifically changed, the text will appear in only one size at the predetermined screen location. This caused the GL text to be oversized at far viewpoints and undersized at nearer viewpoints. As a result of these characteristics, the viewer sees text sinking and disappearing into objects that are moving in the viewing area. From some vantage points, the text on the HUD and radar would obscure other critical indications and the instrument became unreadable. To solve these problems a query was sent to other offices on base that develop graphics applications to see if other graphics efforts were attempting to solve the same visual problems concerning an adequate 3-D text display.

4.2.2 Wright Lab 2D text (Graphtext/Graphfont).

The Wright Laboratories responded to the request for scaled text by supplying their stroke vector text that draws a specified text font using the GL line drawing routines. This text consists of two C++ classes called Graphtext and Graphfont. Graphfont is the two dimensional representation of line segments describing the font of the text. Graphtext is the set of necessary subroutines which allow rotations around the screen in the x-y plane, scaling, and color control. Since the text font is described in two dimensional space, it is easy to modify the code to put a third depth variable in the font description so that the text is then movable in all three dimensions. This allowed the vector drawing of text to remain scaled and oriented in the proper position for any specified view point. This greatly enhances the realistic appearance of the instrumented displays.

The other features of Graphtext allow runtime scaling and color changes to the text. Various sized lettering on displays allows increased accuracy in modelling. The ability to change the standard background polygon of the Graphtext is useful when real time color changes are needed as warning indicators on an instrument.

4.3 Arbitrating Input Devices

Each aircraft type has a unique throttle and stick switch configuration used to manipulate the different aircraft systems. Software updates to actual aircraft systems can often lead to differing functions for the same switches in different models of the same aircraft type. There is no standard control input configuration except possibly for the stick's trim hat, trigger, and weapons release button. The virtual cockpit faces its own switch limitations with the game quality THRUSTMASTER hands-on throttle and stick (HOTAS) (figure 11).

The THRUSTMASTER throttle quadrant has seven available buttons, one rocker switch, one trim wheel, and one two-dimensional trackball. The stick has the trim hat, three buttons, and a trigger. The HOTAS is provided with supporting software documentation, but lacks a detailed button layout (4). One HOTAS hat function, `hat_cam`, is not reliable. As a result of this unreliability, the `hat_cam` data position in the HOTAS bit stream is unusable. The ability to use the rocker switch to identify two or even three switch configuration modes can double or triple the available switch combinations for growth of the VC systems. For a simple generic aircraft, the HOTAS should support short term needs.

Figure 11. HOTAS

The main complaint about the HOTAS is the game quality construction which is already showing signs of deterioration. Some erroneous switch activations occur just by movement of the throttle quadrants without activating the physical switch. Additionally the trackball output varies significantly from HOTAS to HOTAS resulting in fatigue and irritation to the user. A detailed description of the HOTAS switch functions is found in Appendix A. THRUSTMASTER has responded to this critique about quality and manufactured several cast aluminum HOTASs for AFIT. These have held up better to constant use, but minor variations in switch function make the new HOTAS respond slightly differently to previous HOTAS switch activations. Most noticeable are the bottom stick switch's constant "hot" value and the reversal of the trackball inputs for cursor movement.

4.4 MFD Object

The Radar is the first object I designed to display a graphical representation of the orientation of DIS players; each of the players participate in the shared simulation environment. When I began developing the FLIR, the Radar object was encapsulated into a multi-function display (MFD) object with both radar and FLIR capabilities.

4.4.1 Radar Object.

4.4.1.1 User Interface. Two buttons on the HOTAS throttle quadrant allow direct manipulation of the radar presentation. Depressing the selection buttons steps through a pre-computed set of radar ranges and radar modes. The ranges cycle downward in radar scale since the normal SA building technique is to progress from the big picture to the narrow area of interest for weapon implementation. A natural sequence of events progresses from long range detection to ever decreasing target separation until actual engagement near the 10-15 mile range (19:10). Cycling past the shortest radar range resets the cycle to the farthest distance. Radar mode selection follows a similar process repeatedly cycling through ground, air, and navigation radar submodes.

4.4.1.2 Initial Design. The initial rendering and placement of the basic radar display screen did not require any complex computations. A coordinate description specifying the four corners of the radar screen allows quick adjustment of the radar screen in the simulation scene. The coordinates are described in actual aircraft dimensions using the GL/Performer body coordinate system. The Performer body coordinate system is a Cartesian coordinate system originating at the designated model description origin with the y axis forward, the x axis right, and the z axis up. The rendering on the radar display of current DIS network entities from the Playerlist (figure 12) that are visible to the radar requires little computation. The rendering is separated from the floating point intensive computations that generate raw radar screen coordinates. Isolating the computations in one section of code permits locating the MFD in the draw thread and the calculations in the application thread. The calculations transform target world positions to radar screen positions

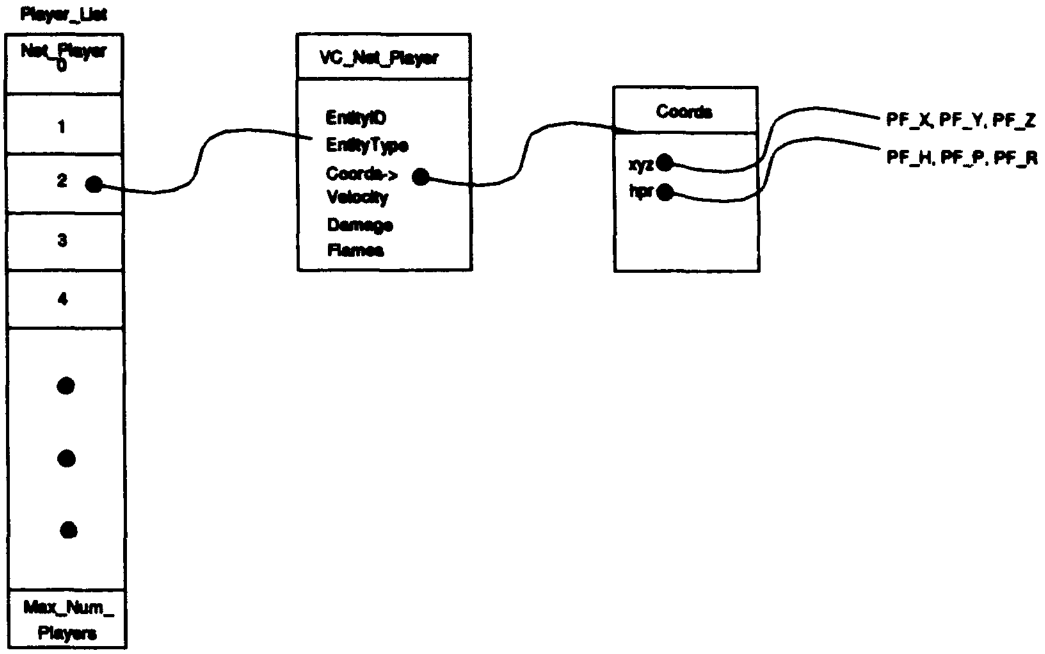


Figure 12. ObjectSim Playerlist

and are passed to the radar. The radar accesses the pre-computed positions and determines the placement of visible players on the radar screen. The computations determining visible radar targets and their corresponding radar coordinates evolved into the INS object that handles the majority of the computations for all radar and FLIR functions.

4.4.1.3 Final Design. For reasons stated in Chapter III, the primary development of the radar was to create the display of a usable air-to-air radar mode. Air-to-ground and navigation modes are presented on the radar with minimal information consisting of relative target position and a selected navigation point in the navigation mode.

Initialization of the MFD specifies the size and position of the radar display in the VC from the four three-dimensional corner descriptions defined in the MFD initialization procedure. All subsequent radar object clipping and position determinations are derived from these four points. Therefore, the radar can be moved to different MFD positions or can be adjusted to fit different aircraft cockpit configurations.

From interviews with air-to-air pilots and during demonstrations of the 89^{th}'s air-to-air task training simulator, the basic requirements for display and necessary computational requirements were developed. A variable scale display is desired allowing 10, 20, 40, 80, and 160 mile ranges in navigation, air, and ground modes. Variables computed for a selected target to aid SA are altitude, magnetic ground track, airspeed, aspect angle, and closure. Aids to identify the target are relative bearing and range from a common steer-point/bulls-eye. Target specific information is only computed or displayed if a particular target is locked-on.

VC 2.0 has a B-scope presentation radar display with a roll stabilized bore-sight orientation. This corresponds to a plan-view of the volume around a line projected along the longitudinal axis of the aircraft forward of the nose. When looking at the radar screen, the observer's position is the bottom center of the display with the top of the display representing the farthest the radar can see for the range selected.

The radar itself is a set of drawing routines that accepts data from the INS and turns the data into a visual representation. Current radar system state is copied into the radar from shared memory that tells the radar which mode to draw, which target information to display, and the number and position of visible radar objects.

The necessary information is in two arrays called the `tgt_LOS_list` and the `tgt_list`. The `tgt_LOS_list` identifies targets visible to the radar while the `tgt_list` provides the radar screen with x and y values of each visibly illuminated object. The x and y radar screen values are not the actual screen coordinates, but the proportionate frustum position the illuminated network entity occupies in the radar frustum. These proportionate values are a percentage of the radar frustum's width and depth. Width (the x value) corresponds to the off-center position the object occupies in the radar frustum. Depth (the y value) corresponds to the depth position the object occupies in the radar frustum. As an example, suppose an air radar 80 mile display

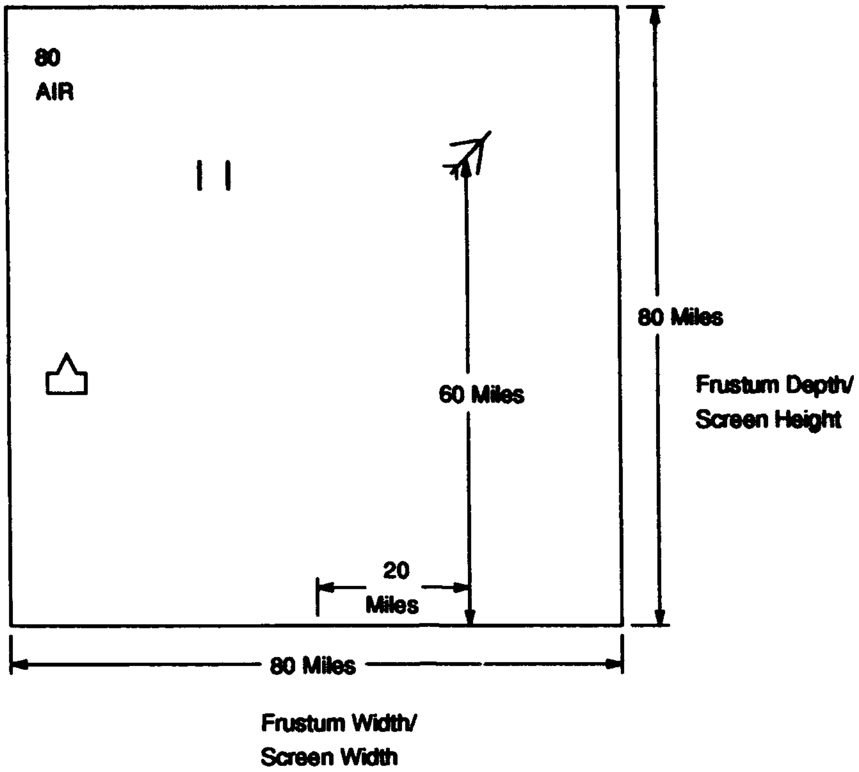


Figure 13. Sample INS To Radar Conversion

is selected and a target is located 60 miles down-scope and 20 miles off-center to the right (figure 13). The `tgt_LOS_list` value is TRUE and the screen percentages are .25 width and .75 depth. These values are multiplied in the radar by the screen width. Screen width is used since the radar display is assumed square. The resulting width value is added to the radar screen horizontal midpoint value, and the frustum depth/radar screen height value is added to the radar screen bottom value. If the radar screen happens to be centered in the cockpit with the bottom edge at 2 meters above the aircraft center, then the horizontal midpoint is 0 and the bottom value is 2. If the radar screen is .1 meter per side, then the target is located at 0.025, 2.075. The radar screen is located $-.05$ to $+.05$ horizontally and 2.0 to 2.1 vertically which describes the radar screen "window".

After frustum width and depth are converted to radar screen coordinates, then they are clipped to the radar screen window. The final clipping determines the illuminated object's icon visibility on the radar screen. These final computations of multiplying coordinate percentages and clipping are accomplished on the draw thread since the INS would not know if multiple radar screens are using the same target list information, or if the radar screens are all the same or differing sizes. The only assumption the INS makes about radar screen dimensions is that a radar screen is square. During each graphical rendering for a frame, the radar state information is updated from the INS information, so the radar refresh rate is in lockstep with the workstation frame rate. A drawback to having the final screen coordinate transformations on the draw thread is the possibility of having numerous objects visible in the display at one time. Given enough objects, these computations may have an impact on the frame rate.

The DIS standard requires that all radar emitters send out state information in the form of an emission PDU if the radar is currently active. In the draft version of the available DIS standard, the specifics of emission PDUs are not fully elaborated. Without a full definition of emission characteristics, the facility for handling emission PDU's was not completed for the DIS network manager. Since the VC could not transmit a valid emission PDU, only the primary procedural calls are in the radar code to prepare to generate an emission PDU if the radar is active. No emission PDUs are actually formatted for transmission.

Depending on the selected radar mode, various displayed radar object icons are visible. These icons include a distinct bulls-eye for orientation information, rectangular targets to show relative position, the radar cursor allowing selective targeting, and sequence navigation points signifying the next route point. In addition to icons, text is also displayed for a selected target informing the operator of a specific target's actions. The cursor is directly manipulable by the VC operator and is therefore dynamic in the radar view. The dynamic characteristics of the cursor caused it to become an object of its own.

4.4.2 FLIR Object. The FLIR is the alternate visual display projected by the MFD. It reflects the view oriented from a simulated IR pod mounted on the underbelly of the VC. This view is weapon dependent and can be attached to an in-flight EO munition and directed along the longitudinal axis of the guided munition.

The FLIR view has two magnifications; normal magnification and four times magnification. When the FLIR is operated in its target designation mode the FLIR orients its view with a LOS vector from its underbelly location and directed towards the target. FLIR roll remains constant at 0 degrees. This provides the desired roll stabilized effect minimizing operator disorientation. Entering CUE mode allows the viewer to directly manipulate the view through cursor inputs which offset the view. After the view is offset, a Performer segment is projected in the direction of the new view orientation. Where the segment intersects the terrain designates the new target coordinates. The new target coordinates are then maintained as the current look position for FLIR orientation.

4.5 Cursor Object

The radar and FLIR cursors are objects the VC operator can manipulate allowing selections of targets on the radar screen or updates to the selected view orientation in the FLIR.

4.5.1 User Interface. Radar cursors move from side to side and top to bottom across the face of the radar screen. FLIR views are also manipulated to pan horizontally or vertically across the designated view area. The HOTAS trackball is used for cursor placement and FLIR movement since the trackball inputs naturally map to a two-dimensional movement. Locking on and breaking lock with a particular radar target is accomplished by depressing a HOTAS throttle button. Repeated depressions of the designate button will cyclically lock and break lock on the designated target.

4.5.2 Accuracy. The HOTAS trackball outputs a range of values dependent on the trackball's rate of rotation. I can take advantage of the rate of trackball rotation and relate the magnitude of the HOTAS data to cursor displacement across the radar screen. Two speeds of cursor travel are created by dividing the range of HOTAS trackball data into two ranges. Low speed associates to the values at the end of the HOTAS trackball range and high speed to all remaining values. The speeds are mapped to incremental distances that are added or subtracted to the current cursor position on the screen. Updates are made each frame cycle allowing fluid movement of the cursor across the screen. Two speed cursor control reduces some of the fatigue in manipulating the cursor since slow adjustments are mapped to small cursor position changes and fast adjustments cause the cursor to move larger distances.

The cursor is active also in the FLIR view and allows updates by reading the trackball in the same manner as the radar cursor. The difference in manipulating the FLIR cursor is apparent when moving the FLIR cursor icon across the FLIR view, the cursor icon remains fixed in the center of the FLIR view while the FLIR view pans across the view area corresponding to the trackball inputs.

The radar cursor and FLIR views move in either one or five unit increments depending on the speed of rotation applied to the trackball. Some HOTAS are very erratic when reading inputs which required mapping a single value to small movements and the remaining values to large movements. One HOTAS refused to produce outputs in the vertical and was therefore unusable. The inconsistencies in reading the HOTAS values limited the gradations available in the cursor travel and FLIR orientation. Precise movements are desired for the cursor since the cursor in the radar and FLIR modes must be placed over specific targets to allow accurate designation in radar sorting and precise targeting for laser designation and electro-optical (EO) weapon guidance.

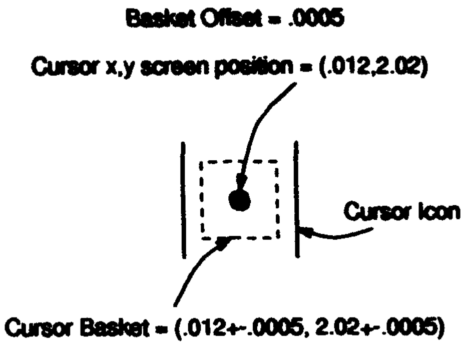

Basket Offset = .0005
Cursor x,y screen position = (.012,2.02)
Cursor Icon
Cursor Basket = (.012+-.0005, 2.02+-.0005)

Figure 14. Cursor Showing Surrounding Basket

4.5.3 Cursor Designation. Cursor designation or selection of a target is accomplished when the cursor is placed over the desired target object icon on the radar display and the designate button of the HOTAS is depressed. A "basket" surrounding the cursor radar screen coordinates is defined from the cursor's x and y screen position and an offset surrounding the position (figure 14). The cursor searches the `tgt_list` to find a visible target whose radar screen coordinates are within the boundaries of the cursor basket. The first target whose coordinates fall within the cursor basket becomes the designated target and a pointer to the current target list is copied into the current air target. This allows the INS to update information to the various weapon and display systems. If no target falls within the basket boundaries, the target closest to the VC and visible to the radar is then designated as the target object.

The cursor will automatically break the radar lock when a not visible indication is determined from a query of the `tgt_LOS_list`. Depressing the designate switch while currently locked on to a target will cause the cursor to interpret the radar lock-on signal as a break lock indication causing the cursor to break the current lock and prepare for another pick of a radar target.

4.6 INS Object

An INS in an avionics system is a device that typically handles navigation tasks to accurately compute current location from sensed position changes. The INS for the VC is an INS in name only since all aircraft position determination is computed in the VC aerodynamic model.

The VC INS performs rudimentary navigation functions by solving time and distance equations to selected navigation points. The results are passed to the navigation panel display. The main computations for the INS involve computing target information as an aid for increasing SA and supplying information to the weapons controller. The VC INS is then a general purpose computational engine handling the VC avionics system, navigation tasks, and elementary targeting functions. Combining the computations in one module simplifies passing state information between different process threads through shared memory variables. The INS handles all significant floating point and trigonometric calculations for the radar, FLIR, cursor, and navigation panel. The greatest number of calculations are spent determining position and orientation of potential targets, and then scaling and orienting the target information for MFD display.

4.7 INS Calculations

The INS calculations are applied to the local navigation reference frame of the VC. The local navigation reference frame is essentially a flat, Cartesian-coordinate system with the x-y plane corresponding to the ground and the z axis corresponding to vertical displacement from the surface. A detailed description of converting to this local reference frame from the WGS 84 reference frame is given in Chapter VI. The local navigation reference frame simplifies the computations of the necessary information to display DIS player information in the VC.

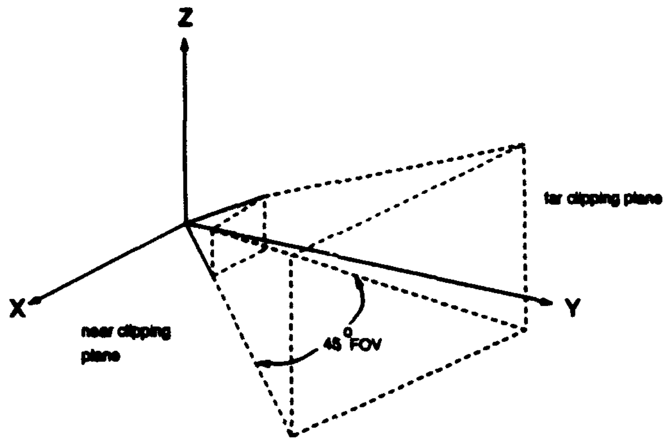


Figure 15. Performer Version 1.2 Simple Frustum

4.7.1 Radar Screen Coordinates. The radar should only display players who could be in the radar's line of sight (LOS). A radar's beam width describes the area of surveillance offset from the radar transmission centerline. Radar range describes the detection distance of the radar. Antenna size and power will limit the radar's detection abilities. Directional radars can either be bore-sighted along the aircraft's longitudinal axis or offset by a limited number of degrees. Search radars typically have a wide beam width while target tracking radars focus the beam narrower to allow more powerful and accurate target illumination. The radar illumination beam clearly corresponds to a viewing frustum used in photography and computer graphics to define view volumes. The VC radar's illumination volume is mapped to a viewing frustum volume inside of which the first iteration of target visibility is determined.

The Performer application programming interface (API) allows the creation of arbitrary viewing frustums. The frustum is built defining the volume specifying the nearest visible distance to the observer, the farthest visible distance from the observer, and the total field of view visible between these distances. The Performer 1.2 simple frustum is initially oriented to view along the y axis which evenly splits the horizontal and vertical field of view (FOV) (figure 15) (34). The simple frustum is then oriented to face in the same azimuth and elevation as the VC.

The INS then has a viewing volume positioned at the origin, directed in space with the same orientation as the VC with the exception of the roll orientation. Two actions are taken as preprocessing steps to the visibility test. First, all objects are tested to see if they would be visible to the particular radar mode selected. As an example, life forms should not propagate down the radar visibility pipeline. Second, the VC's position is subtracted from all Player positions resulting in coordinates that are now in the same reference frame as the oriented frustum. The Performer `pfPtInFrust` call is applied to all the translated points to see if any points are in the frustum. If a point is within the frustum, then further visibility checks are done. The Player may be in a viewing frustum, but it could still not be visible to an actual radar beam if the object is obscured by some other feature in the environment such as terrain, or some other player between the frustum sampled player and the radar.

Testing for obstructions to the LOS in the viewing frustum is accomplished by using the Performer Segment structures (figure 16). A line segment is constructed between the VC position and the visible player. Accurate modelling of true LOS is accomplished with segment intersection testing.

Performer segments are built between each frustum-visible player and the VC. Segment intersection is then applied against the entire modelled scene. When an intersection is detected, the x,y,z coordinate of the intersection point is returned. The distance to the intersected point is compared with the total segment length. If there is a significant difference in lengths, then the Player is considered obscured from the radar by another object and tagged not visible. An intersection *should* always occur since the player should have a model description in the data files which will intersect the segment. An offset distance is used as a discriminator to tell if the intersection occurs against the player itself or some other player or terrain. If the difference in distances between the VC radar and the player is the same or less than the discriminator, the INS assumes there are no obstructions between the radar and the player. No obstructions keeps the `tgt_LOS_list` value TRUE,

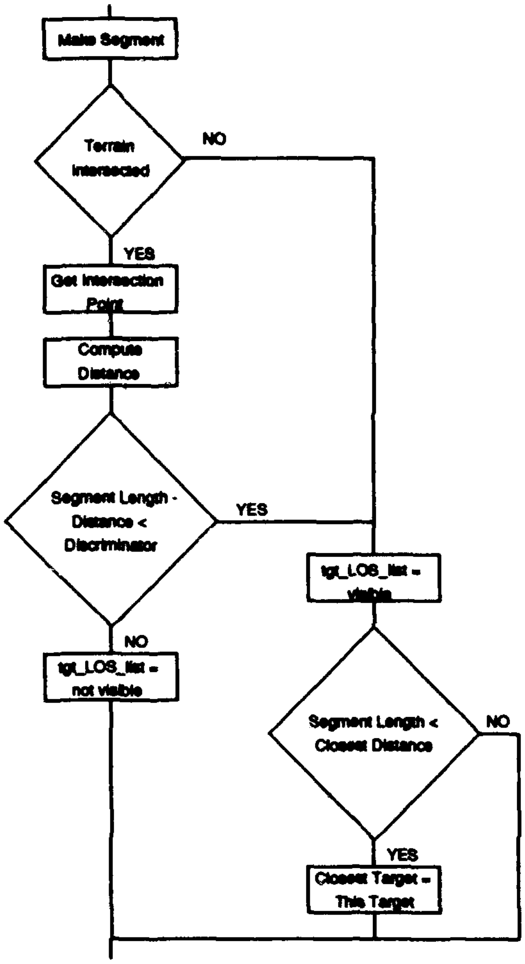
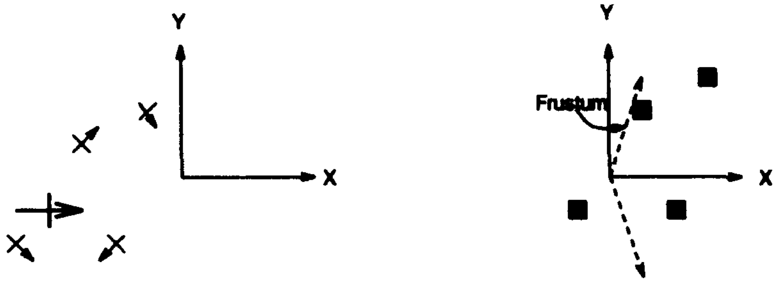

Make Segment
Terrain Intersected
NO
YES
Get Intersection Point
Compute Distance
Segment Length - Distance < Discriminator
YES
tgt_LOS_list = visible
NO
tgt_LOS_list = not visible
Segment Length < Closest Distance
NO
YES
Closest Target = This Target

Figure 16. Logic For Line Of Sight Testing

otherwise the distance difference must be greater than the discriminator and the value is FALSE. A TRUE condition triggers a distance comparison against the closest distance previously computed up to this time. The closest distance player is always tagged so the closest target is always available for radar lock-on.

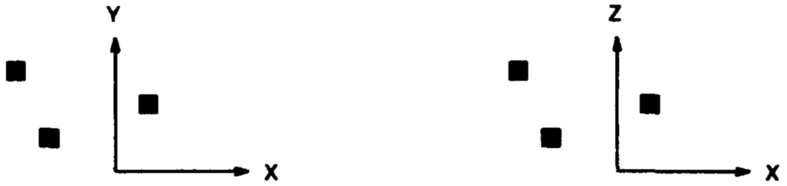
Only after the final true LOS determination occurs do any three-dimensional player coordinates get converted to two-dimensional radar screen coordinates. Figure 17 shows the process applied to Playerlist coordinates. The final calculations are applied to Playerlist coordinates that meet LOS conditions. If the VC was displayed on the radar screen, it would be located at the bottom center of the two-dimensional radar screen, lying flat and directed toward the top of the screen. All potentially visible player coordinates can be translated to a corresponding aircraft origin by simply subtracting the VC coordinates from the player coordinates (figure 17b). The target coordinates of the players visible in the radar frustum then get rotated opposite the VC heading and opposite the VC pitch. These rotations align the targets in a constant forward direction off the VC longitudinal axis corresponding to the center top position of the radar screen (figure 17c). VC roll is not countered since the radar system is ideally roll stabilized and the display should not change with varying degrees of aircraft bank. Initial placement of the horizontal description to the radar screen coordinates is done through a 90 degree rotation around the x-axis followed by a scaling according to the current radar range (figure 17d). The final modification to the player coordinates is a forcing to a specified depth matching the current radar screen position on the y-axis. These transformed coordinates are now loaded into the `rdr_tgt_list`. The `rdr_tgt_list` is then used along with the `tgt_LOS_list` to sort and draw visible radar targets (figure 17e).

4.7.2 Air Target Computations. Air target computations require many floating point trigonometric calculations and are only done in the air radar mode while locked on to a target. Relative bearing computations, from known position to target, formed the basis for target information. This information is used for SA



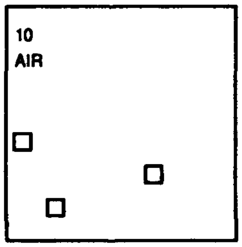
a. Original reference frame orientation, 90 deg hdg, 0 deg pitch, 0 deg roll

b. Translation to aircraft reference frame and radar frustum visibility



c. Rotation opposite aircraft heading and pitch

d. Rotation 90 deg around X axis, range scaling, and y set to 0



e. Convert to screen coordinates, clip, and display

Figure 17. Transforming World Coordinates To Radar Screen Coordinates

building in relation to the bulls-eye, directing views in the FLIR to selected locations, and also for propagating custom built drones to fly preprogrammed navigation routes. Other computational formulas and display requirements were taken from the manual specifying the Display and Debriefing System (DDS) used at Nellis Air Force Base for the Red Flag exercises (10).

4.7.2.1 Steer-point reference. These computations formed the basis for numerous VC tasks. The computations were eventually built into a subroutine called `get_look_angles`. Using the trigonometric properties of the *sin*, *cos*, and *tan* functions with the differences in a reference coordinate subtracted from a target coordinate, the azimuth and elevation angles from the reference to the target coordinate is calculated. These angles can be displayed on the radar as a bearing to a target, used in the FLIR to direct a view in the specified azimuth and elevation, or assigned to a Player's Coord->hpr to direct a flight path or ground track.

4.7.2.2 Closure V_c. The standard closure computations from the Red Flag Measurement And Debriefing System (RFMDS) documentation is used to calculate own-ship versus target closure (10:sheet 3-132).

$$V_c = -(V_{target} - V_{ownship}) \bullet R \tag{1}$$

where

$$R = (Position_{target} - Position_{ownship}) \tag{2}$$

is the normalized LOS vector between the aircraft. This dot product of differences in target velocities and own-ship LOS vector is available since the Player's velocity is part of the DIS standard.

4.7.2.3 Altitude. Player altitude is gleaned directly from the Player Coord->xyz[PF_Z] value. Round earth terrain simulations perturb the altitude value the further the Player is from the local origin. This situation is handled with a subroutine call described in Chapter VI.

4.7.2.4 Aspect Angle. Aspect angle computations for determining the degree of threat from a target are defined from the RFMDS documentation (10:sheet 3-127). The AOT formula used to determine aspect angle is modified to show ownship AOT in relation to the target's tail. The equation then becomes

$$AOT = \pi - \arccos(R \bullet L_{x,target}) \tag{3}$$

where R is the normalized LOS vector and $L_{x,target}$ is the normalized vector comprising the x column of the direction cosine matrix describing the target orientation. This value is returned in degrees and an analysis of the roll orientation of the target determines whether to assign a left or right indication to the AOT value in the radar display.

4.7.3 Ground Target Computations.

4.7.3.1 Range. Range information is determined by the two dimensional projection of the direction vector onto the X-Y plane.

$$Range = |(xyPosition_{target} - xyPosition_{ownship})| \tag{4}$$

This resulting vector gives the straight line ground distance needed for travel to arrive over a selected ground position.

4.7.3.2 Time To Go (TTG). TTG is calculated using the ownship's x-y velocity vector and the *Range* value.

$$TTG = \frac{Range}{xyVelocity_{ownship}} \tag{5}$$

4.7.3.3 Steering. The navigation point coordinates specified in the current INS position determines the directional steering displayed on the HUD. The INS position is updated in the FLIR mode by using CUE and directing the line of sight vector to intersect some other position on the ground. The new updated target information is then used to compute steering.

4.7.4 Network Management. Some disturbing trends appeared in the limited scenarios built for the VC application. INS computational limits were encountered while supporting the SIGGRAPH 93 demonstration in coordination with the Naval Postgraduate School (NPS). The NPS system generates approximately 250 network players, depending on how many bombs and missiles are in flight and the number of actual network participants. The majority of the network players are static objects defining the various cultural and natural features that can react in the simulation. Most of the static players are concentrated in two positions, the airfield and the town south of the airfield. During SIGGRAPH with the VC oriented toward these players in ground or navigation mode, the application thread was slowed down because of the number of computations done on the approximately 200 visible players. The frame rate dropped to under ten fps, and the fidelity of the simulation suffered. Obviously a change had to be made in the computation algorithm. The solution to the problem was to subdivide the computations accomplished on the current player list.

The ObjectSim VC network manager subdivides the work done on the active network players. This action was taken anticipating the large scale simulations that could entail thousands of participants. I did not anticipate the concentration of players that might be visible in the radar search volume since I assumed the search volume to be relatively small compared to the total simulation area. A subdivision of computations that mimics the method of subdivision in the network manager allows the radar to remain updated and not slow VC frame-rate.

Subdivision is accomplished by systematically traversing the player list by a specified modulus. The standard modulus set in the VC INS is ten. This allows an update on every player every tenth graphics frame. During the first cycle the 0^{th}, 10^{th}, 20^{th}, $\cdots$, $\leq$ *max_num_players* in the Playerlist are updated. The second cycle updates the 1^{st}, 11^{th}, 21^{st}, $\cdots$, $\leq$ *max_num_players* of the Playerlist and so on until the modulus is reached, at which point the sequence is started over again.

The ramifications of this modulus computation are that for every rendered graphics frame, only one tenth of the possible total players will be updated to the `tgt_LOS_list` and `rdr_tgt_list`. This results in a gradual filling of the radar screen with visible players and then a clearing of the screen when the modulus is reached. The solution to this reduced radar memory is to build a LOS and target position list that serves as the history for the last complete subdivision sequence. When the radar arrives at the subdivided modulus, the `tgt_LOS_list` and `rdr_tgt_lists` are copied to cleared out memory lists and the current lists are then cleared. In this way the last positions of the visible targets can be displayed if the current and memory lists are *both* drawn on the radar screen. No information is lost and modifications to the cursor designate routine allow for cursor designation only after a complete current `rdr_tgt_list` is built.

The subdivision modification allowed the ground and navigation modes of the radar to run the VC at the same frame rate the VC ran with the radar in standby mode (15 to 20 frames per second) before the subdivision. The subdivision is not implemented while the radar is in air mode since the possibility of having hundreds of air targets on the fighter's radar display would be a rare occurrence. If this is not the case in future simulations, then the subdivision can be applied to all radar modes.

This development provided the basis for a stand-alone interactive VC in a network environment. To aid development of weapons effects and accurate player interaction, along with providing an interesting controllable simulation environment, various other capabilities were added to the VC. Some of these capabilities were developed into applications of their own. The most prominent enhancements were the SAFOR application, the moving parts class, and the experiments on the ARPA miniDART.

V. Development Aids And Simulation Environment Enhancements

During software development, a set of simulation aids and enhancements evolved that were used to validate interaction with the simulation. This set also provides an interesting simulation environment to any user that is operating the VC without the benefit of other active network players.

5.1 Moving parts

The DIS standard provides the means of describing the movements of articulated parts. An articulated part is some part on the body of a simulation entity that can move relative to its other body parts. The usefulness of an articulated parts description is apparent when viewing a tank or other weapon system with moving surfaces. Identifying the orientation of a muzzle is critical when determining the time available for reaction to a threat. An air vehicle may not have many articulated parts unless the aircraft is a tilt-wing such as the V-21 Osprey, or a large weapon platform such as the AC-130 Spectre. Experimental aircraft are displaying further characteristics where an articulated part would be crucial in determining action. High AOA canard designs, vectored thrust nozzles, and air-brakes will need to signal their movement as an articulated part in a high fidelity simulated air-to-air engagement. The movement of each of these parts communicates aircraft state useful in building SA.

The Moving_Parts class built for the VC allows operator interaction with a defined moving part. Two sets of moving parts are in the VC: the F-15 air-brake and landing gear. This class takes a model of the appropriate moving part, places it in the correct position on the aircraft, and moves the part to an alternate position when commanded to do so. Orientation of the part is available from the part's current Performer Coords definition. The orientation is then used in displaying the moving part's position on a cockpit warning panel in the VC.

5.2 Drones and Ducks

The drones and ducks became simulation entities that would react to detonation PDU's to test accurate reception of weapons effects while also serving as test-beds for model switching and weapons effects sequences. These simulation entities allowed cross checking of reference frame orientation and simplified the location of errors in the orientation of model representations within the VC simulation environment. The duck's consistent, programmable routes allowed repeated testing of weapons systems and weapons effects against a predictable entity. This capability supported accurate and quick debugging of simulated weapons system code.

During SIGRAPH 93 these simulation entities provided an interesting environment filling the simulation with players that flew as a participant in the virtual environment. This allowed users of the VC to interact with other network players even when a live participant was not available to operate other distributed network interfaces.

The ducks became the basis of a local semi-autonomous forces (SAFOR) class which allows modifiable force sizes and kinds of network players. This SAFOR class is used to exercise the network bandwidth and the capability of the other graphics applications to operate with numerous network entities. The maximum number of network entities broadcast locally on the graphics labs local area network is 2000 drones running from four separate SAFOR programs of 500 entities each.

5.3 Timers and Reconstruction

In anticipation of the large numbers of people that would be interested in trying out the VC at SIGRAPH, a timer to limit user action while displaying the time remaining in the demonstration was built. This interface also allowed the reconstruction of the simulation environment by rebuilding the landscape so all models of local network players in the simulation which were in a destroyed state could be returned to their original state.

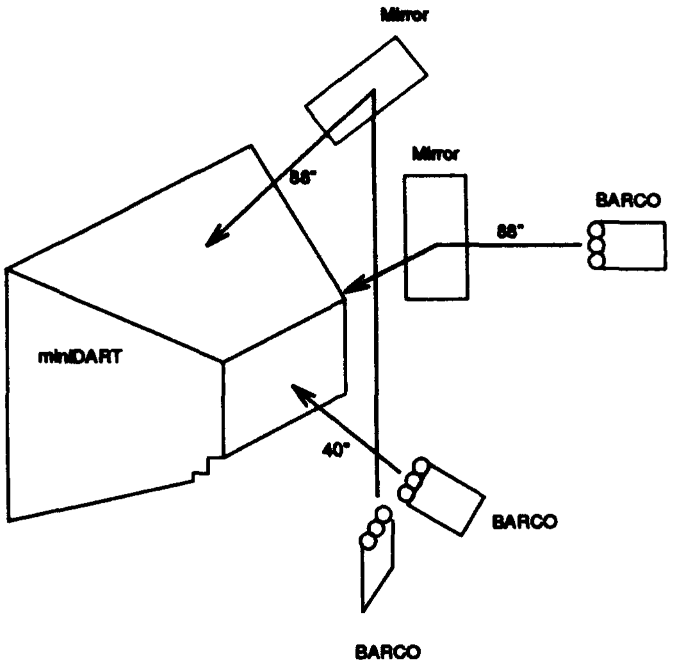


Figure 18. miniDART External View

5.4 Mini Dart

The VC was taken to the ARPA simulation center to install the program on the center's 16 processor Onyx. This effort determined if the VC would display on multiple graphics channels with a single graphics pipeline driving multiple displays. The miniDART is a display device using back-lit BARCO projectors to give an unlimited FOV and 360 degree panorama similar to a domed simulator, but without the curvature and corresponding luminosity projection problems (figure 18). This display was created by Armstrong Laboratories as a derivation of the original Display for Advanced Research and Training (DART) investigating low cost, large field of regard display devices (43).

The miniDART required a configuration modification since the miniDART's projection screens are supposed to enclose a mock-up of an aircraft cockpit. The cockpit mockup is no longer available and the VC also has no method to drive actual instrumentation. To solve this problem, one channel of the graphics pipeline

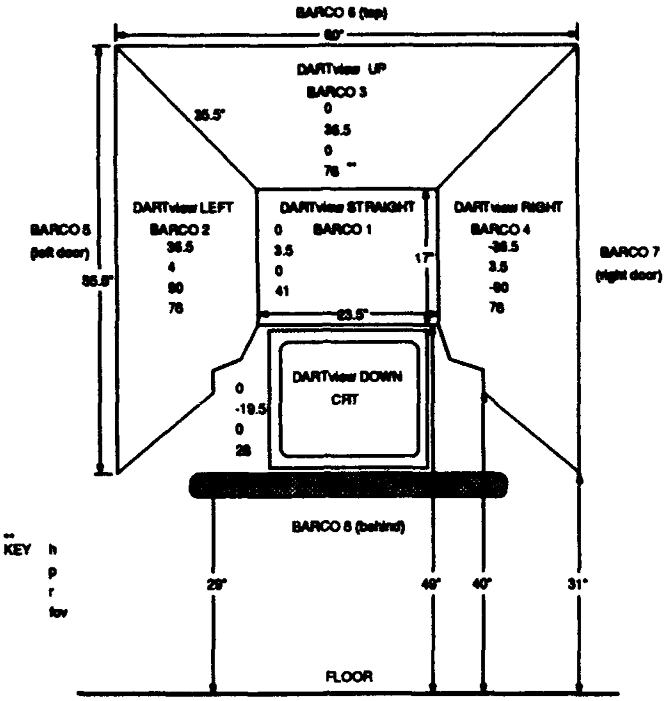


Figure 19. miniDART Projection Surfaces

is directed to a CRT placed under the main viewing section of the miniDART to create a "glass" cockpit of the VC graphics based instrumentation (figure 19). The VC on the miniDART uses a new view class built for ObjectSim that allows five channels within the third graphics pipe of the ARPA simulation center Onyx to project the required displays. This display results in 180 degrees horizontal FOV with approximately 120 degrees FOV from the vertical position to the bottom of the instrument display.

5.5 Round Earth Terrain Model Development

The graphics lab has a program which will transform DTED earth surface descriptions to a curved surface format. The resulting terrain model is positioned in the correct angular orientation corresponding to that part of the earth in an earth-centered earth-fixed (ECEF) coordinate reference frame. The single precision floating point limitations of the Multigen modeler result in the terrain position being

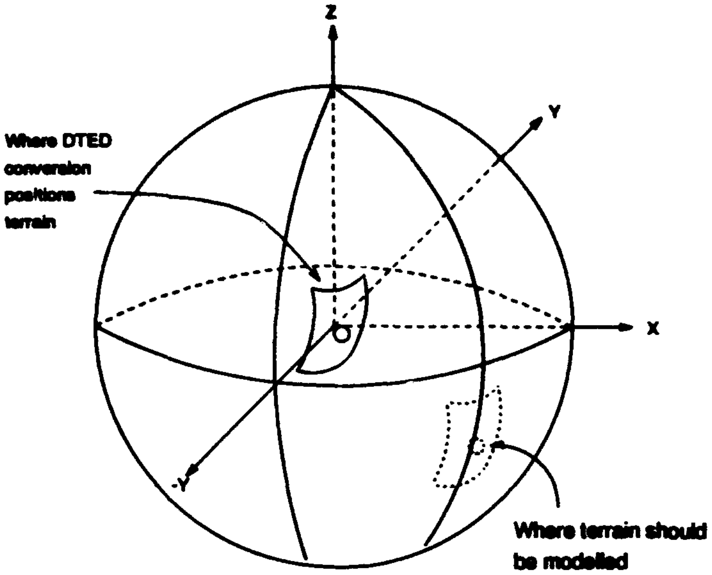

Z
Y
Where DTED conversion positions terrain
X
-Y
Where terrain should be modelled

Figure 20. DTED Terrain Transformation Routine Results

described in x, y, and z values that are centered around the origin of the ECEF reference frame rather than at the actual surface position (figure 20). Multigen will not handle values accurately that are in excess of one million meters and therefore lacks the capability to model polygons that are full earth size. An output from the terrain transformation routine is an x, y, z coordinate tuple which describes the translation required to move the newly oriented terrain into the correct position in the ECEF reference frame. This resulting terrain model and translation tuple is used in the round earth utility class to orient modelled round earth terrain in all AFIT graphics lab DIS applications. A complete description of the round earth utilities is in the following section.

VI. Implementing Round Earth

6.1 Introduction

The AFIT Virtual Cockpit (VC) research project was developed using a flat earth orientation originally implemented for SIMNET (42). An existing flat earth aerodynamic model was used and weapons applications were added that could function in this flat orientation (7)(12)(16). The first attempt to conform to the DIS standard was an implementation of the existing flat earth VC to handle the new protocol data units. DIS is standardized to use a round earth position and orientation description (18:3). When the VC code was modified to attempt implementing the round earth standard in Performer using the round earth orientations, the VC encountered some problems. The problems related to the flat orientation aerodynamics and ballistic computations along with the flat orientation rendering given by Performer. The following is a description of the identified problems, and the methods developed to overcome these obstacles.

6.2 Coordinate Systems/Reference Frames

Several reference frames (coordinate systems) are used in engineering applications. They are all right-hand Cartesian coordinate frames consisting of mutually perpendicular x, y, and z axes. The axes differ in the locations of their origins, the orientations of their axes, and their relative motion between reference frames (Table 3). Column two in table 3 describes if the reference frame appears fixed or moving in relation to the viewer's perspective. In the VC, the view orientation is primarily in relation to the eye position in the aircraft cockpit. The only other common user views in the VC are through the underbelly position of the FLIR pod or as an attached view to electro-optic (EO) guided weapons.

In addition to the various engineering coordinate systems, particular applications such as the SGI GL (22) or Performer (23) libraries will define their own specific reference frames to provide orientation in the viewing volume of interest. Each reference frame that impacts the VC is explained in the following sections.

6.2.1 GL/Performer Reference Frames. The flat earth Performer interface works well with simple, relatively small areas since these terrain models correspond to the Performer coordinate system (Figure 21) (23:4-8). The geometric model of the terrain can be oriented with the z axis up, y axis north, and x axis east. Performer always considers Z up and orients the Performer EarthSky model (clouds, horizon, fog, time of day and other environmental features) to this Z up orientation. The built-in Performer features provide an easy way to build a detailed simulation without spending a great amount of effort trying to model these effects on your own. The problem with the performer application is that the built in features of the EarthSky model are not flexible in their orientation and can never be placed on a tangent to the surface of a sphere. This flat earth orientation has significant drawbacks if the simulation is required to operate in a round earth or spherical terrain description.

6.2.1.1 Body Coordinates. The Performer body coordinates are a non-standard system when compared to an aerodynamic definition of body coordinates (section 6.2.5). Performer orients an object with the x axis out the front, the y axis out the right side, and the z axis up (Figure 22) (23:4-9). Performer body coordinate descriptions are maintained in a coordinate structure called Coords which

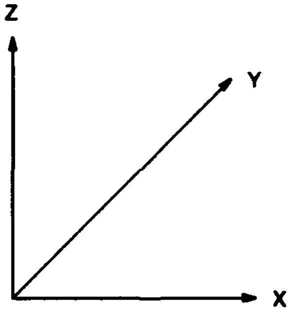


Figure 21. Performer Coordinate System

Frames		
Coord Sys	Fixed/Moving	Origin Loc.
GL Body	Moving	Body C.O.M.*
GL Screen	Fixed	Lower left corner
GL Frustum	Fixed	Eye Position
ECEF	Fixed	Earth C.O.M.
ECI	Fixed	Earth C.O.M.
n frame	Fixed	Local Origin
b frame	Moving	Body C.O.M.
*Center of mass (21)(23)		

Table 3. Reference Frames.

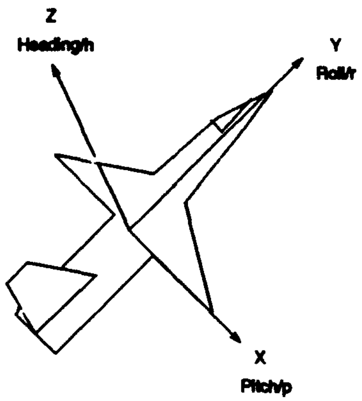


Figure 22. Performer Body Coordinate System

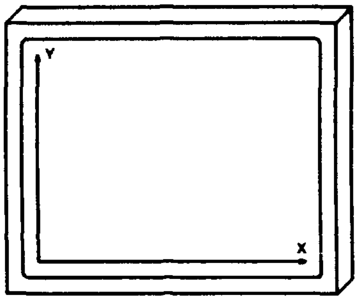


Figure 23. Graphics Library Screen Coordinates

points to x, y, and z positions, and heading, pitch, and roll (h, p, r) body angular orientations. Performer implements its own version of Euler angles (section 6.3.2) and uses these to create the body rotations used in the Coords->hpr, for the objects dynamic coordinate description. Heading, pitch and roll information is easily retrieved from the Coords->hpr but Performer does not follow the standard aerodynamic or graphics manipulation of a body coordinate system by a set of Euler angles since the Performer body coordinate system is different. The difference between Performer "Euler" angles and standard Euler angles is discussed later.

6.2.1.2 Screen Coordinates. Direct manipulation of screen coordinate positions from the VC operator point of view is seldom done in the VC application. The screen has its own coordinate system with x as the horizontal screen axis, y as the vertical screen axis, and z perpendicular to the screen being forced to zero, but used as a buffering cue for hidden surface removal (Figure 23) (22:7-3). Some VC functions such as the radar require manipulating three dimensional world coordinates into a two dimensional screen coordinate space for presentation to the operator.

6.2.1.3 Viewing Frustums. A viewing frustum definition is needed to define volumes of interest to aid in culling objects, or visibility detection of weapon systems (Figure 24). When orienting a frustum, the x axis projects right, the y axis projects up, and the z axis projects towards the viewer so the volume is defined in

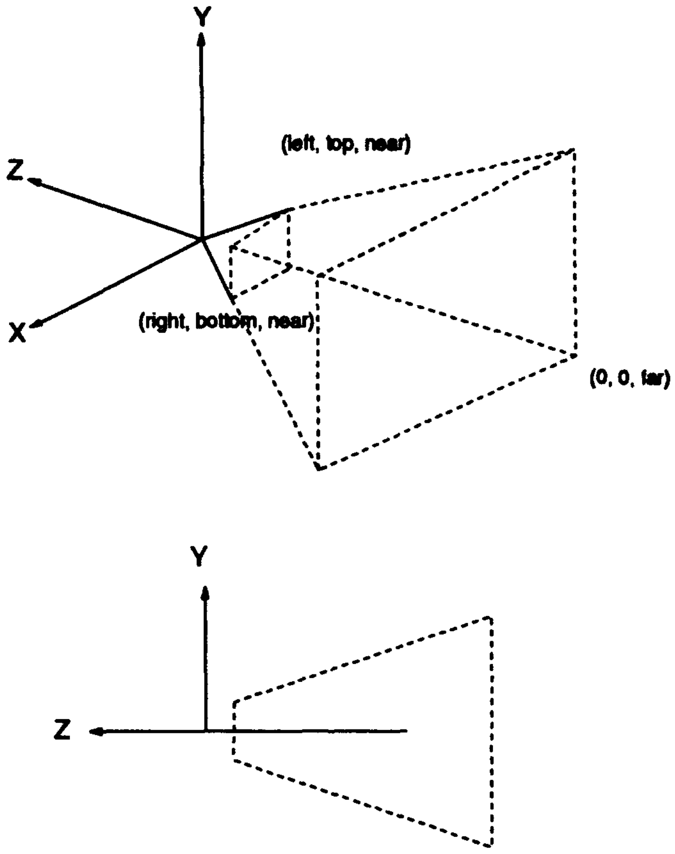


Figure 24. GL Frustum Coordinate System

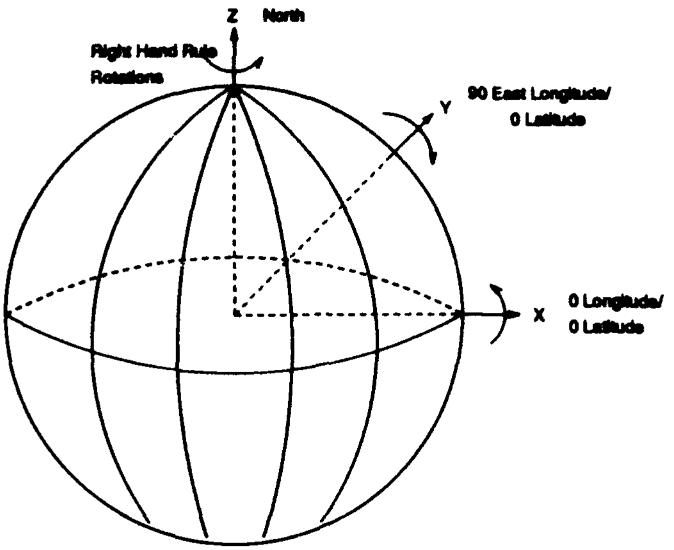


Figure 25. WGS-84 Earth Centered Inertial Coordinate System

the -z direction (22:7-8)(23:4-6). Performer version 1.2 now orients frustums along the y axis away from the viewer, with the x axis right, and the z axis up.

6.2.2 WGS 84/Earth-Centered Earth Fixed (ECEF) Frame. Locations in the simulated round earth are identified using the World Coordinate System. The shape of the world is described by the WGS 84 standard, DMA TR 8350.2 (18:3). The WGS 84 reference frame is shown in Figure 25. This reference frame has its origin at the center-of-mass of the earth. In addition, it has its z-axis aligned with the earth's spin axis passing through the north pole. The x and y axes of this frame are in the earth's equatorial plane and are fixed so they rotate with the earth. The x axis extends through the Greenwich meridian (0 deg longitude), while the y axis extends through the 90 deg east longitude. A distance of one unit measured in world coordinates corresponds to a distance of one meter in the simulated world (18:3)(21:29).

6.2.3 Earth-Centered Inertial (ECI) Frame. The ECI frame is similar to the ECEF since the ECI origin is at the center-of-mass of the earth and the z-axis is aligned with the earth's spin axis. The difference is that while both ECEF and

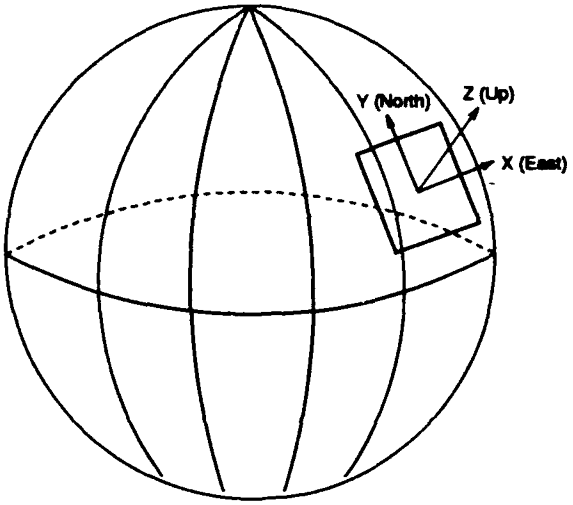


Figure 26. ENV Coordinate System

ECI frames still have their x and y axes in the equatorial plane, the ECI frame does not rotate with the earth spin axis. Therefore the ECI is non-rotating with respect to inertial space (with respect to the *fixed stars*) but the ECI does accelerate with respect to inertial space since it moves with the earth center of mass(21:29). The ECI and ECEF frames are aligned once every 24 hours when the Greenwich meridian aligns with the ECI x-axis.

6.2.4 Navigation Frame (n-frame). The navigation frame has its origin on, or close to the surface of the earth, usually at the location of an inertial navigation system. The x-y plane of the n-frame is tangent to the surface of the earth with the positive x-axis pointing true north, the positive y-axis pointing east, and the positive z-axis pointing "*down*". This is the north-east-down (NED) convention. Another navigation frame used frequently is the east-north-vertical (ENV) representation (figure 26). For the ENV frame, the x-axis points east, the y-axis points north, and the z-axis points up (21:31). The ENV frame will become the local reference frame used in the VC application.

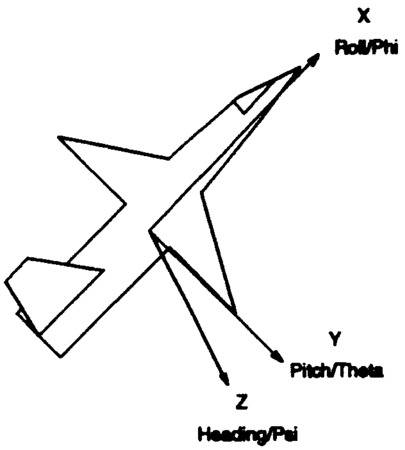


Figure 27. Aerodynamic Coordinate System

6.2.5 Aerodynamic Body Coordinates (b-frame). The body frame is fixed to some rigid body (or vehicle) whose motion is being analyzed. The origin of the *b*-frame is usually located at the center-of-mass of the body, and the axes of the frame have the same rotational motion as the body. When the body is an aircraft, and also for participants using the DIS v2.0.3 standard, it is the convention to let the positive x-axis point out the nose of the aircraft, the positive y-axis point out the right wing, and the z-axis point out the bottom of the aircraft, as shown in Figure 27 (18:4)(21:32).

6.3 Coordinate Transformations

The previous reference frame descriptions show the many, often conflicting ways in which commonly used coordinate systems are described. Since these many reference frames exist, there occurs the need to move from one frame of reference to another. To accomplish the change of reference, it is necessary to transform a mathematical vector "coordinatized" in one frame, to the equivalent vector coordinatized in another frame. The term coordinatized is used meaning the position and orientation description of the object of interest. The standard method of accomplishing this change of reference frame is through a direction cosine matrix (DCM) built from Euler angles (21:32). Once a DCM is built from the Euler angles, the necessary transformation from one reference frame to another is composed from the rotation matrices that will align the world coordinate system with the orientation desired.

6.3.1 Direction Cosine Matrix (DCM). The direction cosine matrix, denoted as C^e_n, transforms (rotates) a vector $\underline{R}^3$ from one frame to another. In this illustration it is from the n-frame to the e-frame. This is done by pre-multiplying the mathematical vector by the appropriate 3 × 3 DCM.

$$\underline{R}^e = C^e_n \underline{R}^n \tag{6}$$

Where C^e_n is the 3 × 3 DCM that transforms a mathematical vector coordinatized in the n-frame (subscript) to the equivalent vector coordinatized in the e-frame (superscript).

$$C^e_i = \begin{bmatrix} C^{x_e}_{x_n} & C^{y_e}_{x_n} & C^{z_e}_{x_n} \\ C^{x_e}_{y_n} & C^{y_e}_{y_n} & C^{z_e}_{y_n} \\ C^{x_e}_{z_n} & C^{y_e}_{z_n} & C^{z_e}_{z_n} \end{bmatrix} = c_{ij} \tag{7}$$

Where each element c_{ij} of the DCM represents the cosine of the angle or a *projection* between the i-th axis of the n-frame and the jth axis of the e-frame (21:33). Methods for deriving DCMs in the general cases can be found in (13:211) by examining the angular differences in the coordinate system axes or through an application of vector cross products (21:33).

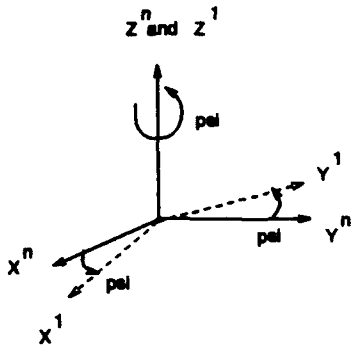
6.3.2 Euler Angles. The *Euler angle* set is another set of parameters which describes the relationship between two coordinate frames and which is useful for attitude displays showing orientation information . The heading of an aircraft is displayed to the pilot on a heading indicator, such as the heading situation indicator (HSI). The pitch and roll of an aircraft are indicated on an "attitude" indicator, such as the attitude directional indicator (ADI). These three angular quantities are the Euler angles (21:37).

Euler angles are not uniquely defined. There is an infinite number of choices for an Euler angle set. For example, a rotation of a body by 90 degrees around the x-axis and then 45 degrees around the newly positioned y-axis will not result in the same orientation as reversing the order of rotations by first rotating 45 degrees around the y-axis and then 90 degrees around the newly positioned x-axis. The first rotation set results in an aircraft in 90 degrees of bank heading 45 degrees while the second rotation results in the aircraft in 45 degrees nose high with 90 degrees of bank with a heading of 0 degrees. As a result of this ambiguity with the same angular values describing different orientations, the particular choice of Euler angles and the rotation order dictated by the application must be applied consistently. If the rotation order of the angles is interchanged, then a *different* Euler angle representation is defined. Of the numerous definitions possible, the most frequently encountered Euler angle definition is that of the aircraft attitude displayed to the pilot (21:40).

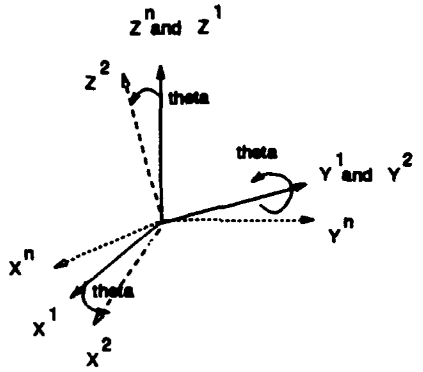
It is convenient to describe the three Euler angle rotations as follows. Begin with the body b-frame and the NED navigation n-frame coincident, that is with the aircraft heading true north, wings level and at constant altitude and airspeed. In this attitude, the three Euler angles heading ψ, pitch θ, and roll ϕ are all zero. From this aligned configuration two intermediate frames are used to define the three rotations (Figure 28).

1. Rotate from x^n, y^n, z^n to x^1, y^1, z^1 about $z^n = z^1$ through ψ.
2. Rotate from x^1, y^1, z^1 to x^2, y^2, z^2 about $y^1 = y^2$ through θ.
3. Rotate from x^2, y^2, z^2 to x^b, y^b, z^b about $x^2 = y^b$ through ϕ.

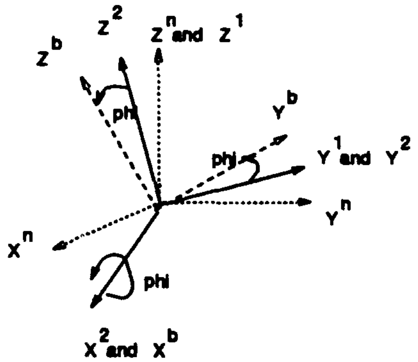
The Euler angle rotations are related to the DCM transformations. Each Euler angle rotation can be represented in the DCM. The complete DCM transformation from



First rotate about z by angle psi



Second, rotate about new y by angle theta



Third, rotate about newest x by angle phi

Figure 28. Euler Angle Body Axis Rotations

$b-$ to $n-$ frame, C_b^n is the product of the pre-multiplication of the individual DCM transformations. This results in the conventional DCM:

$$C_b^n = \begin{bmatrix} \cos\psi\cos\theta & \cos\psi\sin\theta\sin\phi - \sin\psi\cos\phi & \cos\psi\sin\theta\cos\phi + \sin\psi\sin\phi \\ \sin\psi\cos\theta & \sin\psi\sin\theta\sin\phi + \cos\psi\cos\phi & \sin\psi\sin\theta\cos\phi - \cos\psi\sin\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{bmatrix} = c_{ij} \tag{8}$$

If the DCM is provided without direct knowledge of the Euler angle values, then the Euler angles are extracted by the following formulas:

$$\theta = -\sin^{-1} c_{3,1}, (-90\,\mathrm{deg} < \theta < 90\,\mathrm{deg}) \tag{9}$$

$$\phi = \sin^{-1}\frac{c_{3,2}}{\cos\theta}, \cos\theta = (1-(c_{3,1}^2)^2)^{\frac{1}{2}} \tag{10}$$

$$\psi = \sin^{-1}\frac{c_{2,1}}{\cos\theta} \tag{11}$$

The DIS standard follows the Euler angle convention in the previous description (18:5)(21:40).

6.4 Round Earth/Aerodynamic Model Problems

The VC aerodynamic model and ballistic computations operate only in an ENV reference frame. The aerodynamic model does not respond correctly when the local vertical orientation does not correspond to the $+z$-axis. When the VC was initially flown around a spherically modelled terrain section, the movement over the horizon "edge" resulted in increasing velocity and accelerations while the VC was oriented straight and level with the terrain. This is due to the local orientation appearing level, but in actuality the VC is pointing in some direction up or down in the absolute world reference frame (figure 29). Moving the "up" vector away from the $+z$-axis in a local navigation reference frame can not be tolerated unless a new aerodynamic model is found. Since the VC is tied to Performer and the Performer problem needed

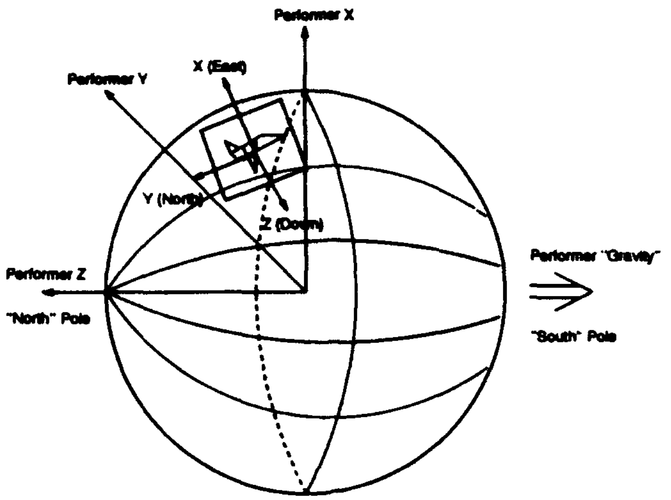


Figure 29. Aircraft Locally Level But Accelerating "Down".

solving, the current aerodynamic model is maintained. The workings of the current aerodynamic model are well understood and modifications to the VC propagation method are avoided by keeping the current model.

6.5 Round Earth/Performer Problems

The transition to the World Coordinate System (WCS) orientation using WGS 84 terrain modelled data reflecting a round earth orientations presented problems with Performer. Any viewer positioned on a round earth does not consider the z-axis through the north pole as *up* anymore in relation to the WCS reference frame, unless the viewer happens to stand directly on the north pole. The *up* orientation is now the normal to the surface of the sphere at the viewer's particular location. A spherical terrain description in Performer is no longer oriented up and level with the Performer X-Y plane except at the x=0, y=0 z=surface position on the sphere.

When Performer attempts to draw the spherically modelled terrain in its rendering routine, the Performer X-Y plane horizontal environmental features are always drawn parallel to the X-Y plane of the WCS. Since the WCS X-Y plane runs through the equator, this can result in environmental cues to the viewer that can be 90 degrees rotated, or upside down, depending on the viewer's physical position on the earth. Terrain located at the equator would be oriented vertically with the Performer horizon and cloud representations perpendicular to the ground. Also, an object's orientation is described in Euler angles in the WCS with placement on the sphere positioned with the object's x, y, and z values. When comparing Performer Coords->hpr with Euler angles, there is no longer a quick method to extract orientation. In Performer, an object's compass direction on the terrain, pitch in relation to the level ground "beneath" the object, or roll in relation to the local horizon was easily extracted from Coords->hpr. In round earth, the Euler angle set must be transformed so this data can be retrieved. This transformation problem is similar to a visual jitter problem encountered in an early version of the VC. The jitter is removed by translating the entire Performer scene graph by values opposite of the view location's x, y, and z values and rendering the scene from the 0,0,0 view location (37:49). Holding the view position constant helps reduce the effect of rounding errors on computations that can make the viewpoint jump within the scene.

A class of round earth utilities was developed that is able to take any geographic location on the earth, designate that location as the local origin of interest, and rotate that local origin systematically to align with the vertical rotation axis of the earth (Appendix B). This rotated position is then translated down to the earth's center. In this way the local origin now acts as the origin of a seemingly "flat" terrain section. The flat terrain then is compatible with the Performer rendering functions and the VC can fly on a reoriented round earth. Specific problems to overcome were numerous.

1. Would this procedure preserve the orientations relative to the local tangent surface ENV reference frame after the transformation? An orientation on the round earth tangent location ECEF must maintain the same orientation in relation to the surface after rotating and moving to the ENV at the "flat" earths "core".

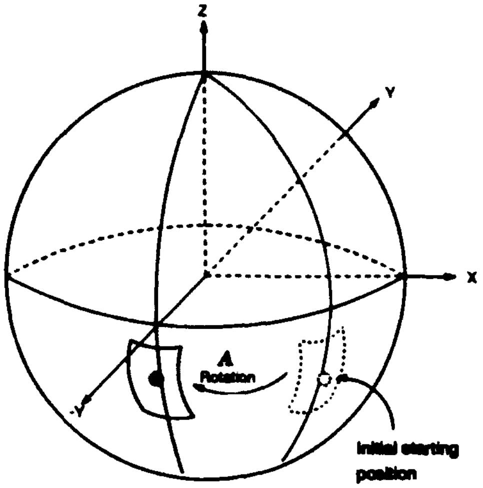


Figure 30. Terrain Rotation To -Y Longitude

2. When the curved terrain is transformed, are altitude values preserved at any given position?
3. How do you determine a local radar altitude at a given position?
4. How can satellite data be incorporated?

Each of these questions were resolved with specific subroutines within the round earth utilities class. The round earth utilities composes the necessary transformation matrices which will transform the coordinates and orientations of interest from one reference frame to another.

6.6 The Round Earth Utilities

6.6.1 Moving The WGS 84 Area of Interest. Moving a WGS 84 terrain patch needs to preserve the cardinal compass directions after completion of the movements. A way to preserve the tangent to the sphere orientation is to first rotate the area of interest in the shortest direction by the angle $\mathcal{A}$ aligning the center of the patch with the earth longitude corresponding with the $-y$-axis (Figure 30).

The second movement rotates the center of the patch by $\mathcal{B}$ to align the center over the vertical rotation axis ($+z$) of the earth (Figure 31). The patch is then trans-

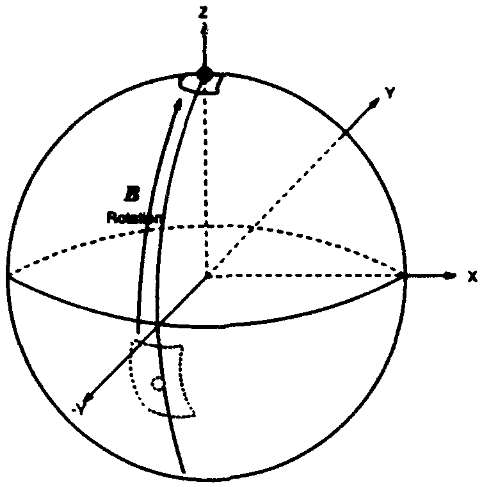


Figure 31. Terrain Rotation To Vertical Axis

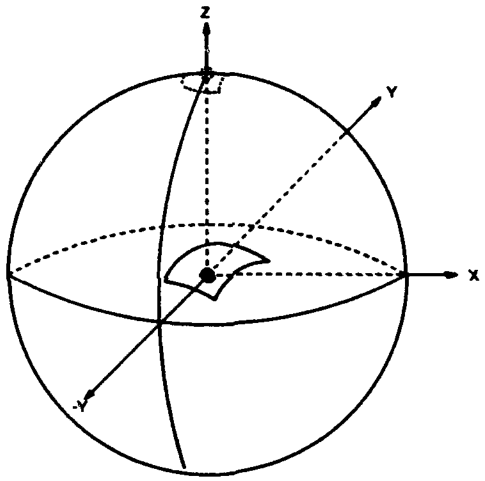


Figure 32. Terrain Translation To Earths "Core"

lated in the $-z$ direction by the distance the original center of the patch of interest was from the center of the earth before any movements were made (Figure 32). This orients the terrain in the ENV reference frame that is suitable for the Performer application and the VC aerodynamics model.

The values to use for the local origin are currently obtained from the extraction of the mean sea level (MSL) coordinates of a specific latitude longitude pair. The latitude and longitude are given to a routine that returns the coordinates for the MSL position at that latitude, longitude point in the WGS 84 data description (38).

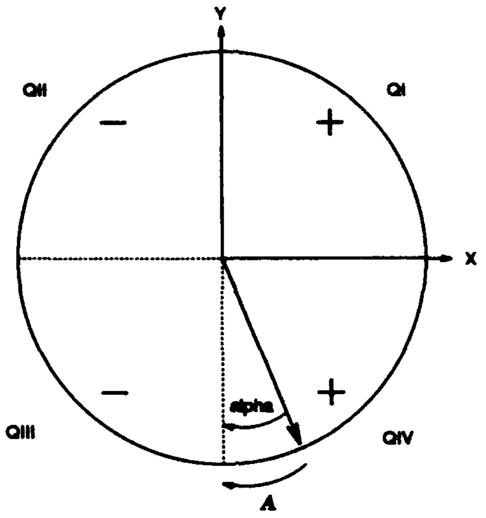


Figure 33. ALPHA Angular Determination

The angle determination is based on the trigonometric properties of the desired angles. α (Equation 12) is the angular displacement from the $\pm y$ axis.

$$\alpha = \sin^{-1}\left(\frac{x}{\sqrt{x^2+y^2}}\right) \tag{12}$$

To determine $\mathcal{A}$, the total rotation and direction by the right hand rule required for alignment, requires determining the quadrant that α is located in (Figure 33). The plan view of the x-y plane shows the right hand rule as positive with counterclockwise rotations. Quadrant determination and total rotation ($\mathcal{A}$) is summarized in equation 13.

$$\begin{array}{lll} QI & \alpha(+) & \mathcal{A} = -(180 - \alpha) \\ QII & \alpha(-) & \mathcal{A} = 180 + \alpha \\ QIII & \alpha(-) & \mathcal{A} = -\alpha \\ QIV & \alpha(+) & \mathcal{A} = -\alpha \end{array} \tag{13}$$

The maximum rotation could be ±180 degrees if the point is originally aligned with the $+y$-axis.

Once the patch is centered along the $-y$-axis, then the vertical rotation by $\mathcal{B}$ is required around the x-axis. Again the angle β is some deflection from an axis (z),

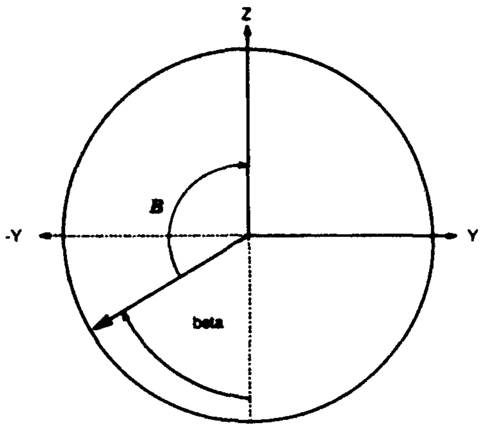


Figure 34. BETA Angular Determination

and the final rotation is always counterclockwise (− rotations for this axis) according to the right hand rule (Equation 14).

$$\beta = \sin^{-1}\left(\frac{\sqrt{x^2+y^2}}{\sqrt{x^2+y^2+z^2}}\right) \tag{14}$$

Equation 14 is always positive, so the angular rotational magnitude must be determined from the sign of the original z value. $\mathcal{B}$ magnitude determination is summarized in equation 15.

$$\begin{array}{ll} +z & \mathcal{B} = -\beta \\ -z & \mathcal{B} = -(180 - \beta) \end{array} \tag{15}$$

$\mathcal{A}$ and $\mathcal{B}$ are the angles used to create the rotation matrices when composing the final matrix (C_r^f) that is applied to both the object's center of mass positions and Euler angles to go from round earth to flat earth. The transpose ($(C_r^f)^{-1} = C_f^r$) is the rotation matrix that can transform the VC's position and orientation in flat earth to the appropriate round earth representation. After rotation and transforming from round-to-flat (rtf), we apply the vertical translation based on the distance from the local origin's MSL position to the center of the earth. When transforming from flat-to-round (ftr), the necessary translations by x, y and z values of the original round earth local origin are applied to place the new ftr position in the proper location *after* the rotations from ftr. This method of direct insertion into the transformation matrix was originally developed to reduce the time for matrix multiplications. Depending on how often the multiplications need to be performed, the utilities can be changed to do a vertical translation before the ftr rotations.

The last step before doing either of the translations is to put the Euler angles into the DCM before application of the transform. Since the aerodynamic coordinate system and the performer body coordinate system are not aligned due to their different definitions, a twist to the performer orientations is required for both the rtf and ftr transforms. This twist is identical to transforming between ENV and NED n-frames. The final composition equations then become for flat earth to round earth:

$$C_{DCMflat} C_{env}^{ned} C_f^r C_f^{rtranslate} \tag{16}$$

The final composition equation for going from round earth to flat earth is:

$$C_{DCMround} C_r^f C_{ned}^{env} C_r^{ftranslate} \tag{17}$$

When creating the Euler DCM, the appropriate sin and cos values of the angles are loaded directly into the DCM according to the transpose of Equation 8. This is done since the Performer `pfMakeEulerMat` routine does not create an Euler rotation matrix that follows the standards for aerodynamics or for DIS. This difference stems from the difference in the definition in body reference frames.

Performer annotates its Euler angles as heading, pitch, and roll. These rotations are applied in the same order as the standard Euler description, but the axis about which these angles are applied are different since the reference frame descriptions of body coordinates is different. In Performer, heading is a rotation about the z-axis, pitch is a rotation about the x-axis, and roll is a rotation about the y-axis.

The DCM produced by these angles for Performer is the product of the rotations about first the z-axis, then the x-axis, and finally the y-axis (23:4-8).

6.6.2 Public Methods For Round Earth Utilities. Public methods for users of the round earth utilities include:

- Defining the round earth local origin which generates all necessary transformation matrices.
- Determining a core segment which will point from an entities current position directly toward where the earth's core (0,0,0) is positioned. When operating in a Performer reference frame the earth's core is some translated value away from 0,0,0 and this segment allows proper "up" and "down" orientation to the local level on the tangent to a curved surface description.
- Computing altitude from a coordinate tuple since z coordinate values no longer relate to an absolute vertical displacement from the tangent surface. This deviation from up becomes more pronounced the further a simulation entity travels from the local flat origin.
- Latitude and longitude extraction from a coordinate tuple to accurately recover location information which is commonly used in navigation for map orientation.
- ECI to ECEF conversion which allows satellite position information to be consistent with the DIS PDU parameters.

This functionality allows all the AFIT graphics lab DIS applications to operate in either a flat earth or round earth terrain description. Initialization is accomplished with a simple `build_terrain` call accessed through the ObjectSim Terrain class. All the necessary DIS PDU transformations are transparent to the user and simple procedural calls are all that is required to guarantee accurate position or orientation information.

VII. Results

The final configuration of the VC consists of an operating multi-mode radar and FLIR with user inputs entered through the HOTAS. The VC operates in many configurations highlighting its varied functionality and easy modification potential. These configurations included:

1. An out-the-window view from an SGI workstation display. View orientation movements are allowed through keyboard inputs or a Dimension 6 spaceball.
2. An immersive environment with the use of a head-mounted display (HMD) deriving view orientation from magnetic sensing of head movement through a Polhemus sensor. Both NTSC resolution (640X480) color HMDs and high-resolution (1024X1024) monochrome HMDs are driven by the VC.
3. An immersive, large FOV, multi-channel, rear projection display applying different fixed viewpoint orientation and FOV for each separate channel (miniDART).

The variety of viewing mechanisms available allows critique of the different viewing mechanisms in relation to immersion. The least restrictive mode of the VC operation is with the miniDART projection display. A large FOV is available even with the limited channels used. Four of the eight available projectors allows the creation of a "half-dome" display. The other four projectors would complete the "dome" effect if driven by the other available channels. This large FOV allows peripheral cues to enhance the feeling of flying. This is accomplished with the rolling horizon at the edge of peripheral vision and relative velocity differences creating "ground rush" with near terrain at low altitudes. Smooth control inputs resulted from the ability to accurately gauge when a turn should start since the pilot can look 90 degrees right, left, and vertically. There is no restriction to the view in the cockpit environment resulting in free access to potential mock-up controls such as landing gear levers, radio switches, or other devices.

The next display in order of acceptability is the HMD. The monochrome high resolution display has advantages over the low resolution due to the need for detail when looking at the instruments. The HUD, radar, FLIR, and cockpit instruments all contain text or small symbols which require pilot interpretation. Without the ability to read the instruments, all potential gain with color realism is lost by the inability to accomplish any useful task. The lack of color in the monochrome display was most apparent at low altitudes when differences in texture color cues aided the sense of relative motion and ground rush. Even though the Polhemus magnetic sensors lagged in tracking head movement, and the projection mechanisms had a limited FOV, the ability to "look before you leap" in leading turns and making decisions allows the pilot to fly much more smoothly.

A major limitation to the HMD is the restriction placed on the pilot when interacting with physical objects used in actual cockpits. If an HMD is to be a success, accurate modelling and extreme sensitivity is required to overlay a virtual image of the cockpit on the actual controls. An alternative is to make all switches, knobs, and dials virtual with user input through a data-glove. This will severely limit the pilots tactile inputs that are crucial when activating cockpit instrumentation. The pilot is maintaining an instrument flying or visual flying cross check of critical instrumentation which rarely encompasses the switches, knobs, and dials that are not critical to safe flight. When activating one of these non-critical switches, the pilot depends on the learned location of the instrument and the feel of the instrument in the hand or under the fingertips to confirm that the instrument finally chosen is the one originally desired. Many switches are designed to enhance this process such as landing light switches with raised bumps, or landing gear levers shaped like tires. Without the tactile inputs, the pilot faces a more difficult task in coordinating accurate inputs in the simulation versus the real flying environment.

The least acceptable display mode is the single out-the-window display offered by the graphics workstation. High resolution monitors do not compensate for limited FOV. If the user is relying on visual cues in the environment outside of the cockpit, the cues needed could come from any angle or position outside of the cockpit. The result with the single view workstation is limited SA since the scope of available visual knowledge is restricted. Also, hesitant control inputs occur since the ability to lead turns and anticipate the position of landmarks or adversaries is hard to determine if they are outside the view of the display.

7.1 Performance Limitations

The major bottleneck in the INS code for the VC 2.0 is the visibility search of detectable radar objects. The selection criteria for visibility attempts to defer any floating point calculations to the very end of the visibility check. Even so, large numbers of objects can still appear in the radar FOV. This is most noticeable with large concentrations of ground objects close together, such as in a town. Visibility culling in the radar frustum FOV greatly decreases the necessary computations. Using Performer 1.1 on a database with 203 entities, the frame rate varied from 10-15 frames per second (fps) looking away from the entities, to 6-7 fps with the radar directed towards the entities. Application of subdivision against the Playerlist allows a partial check of radar visibility against the entire Playerlist. Implementing the subdivision with a modulus of 10 on 203 players resulted in the radar operating at the same rate when directed toward the players as when directed away from the players. When using Performer 1.2, the frame rate went up an additional 5 fps resulting in a sustained VC 2.0 frame rate around 15 fps.

When the FLIR channel operates, the frame rate of the VC drops by 6-7 fps. The radar computations are disabled in the FLIR mode, so the FLIR frame rate is consistent at 10-12 fps. If a user is looking at the FLIR, outside visual cues are not as important to the user as the FLIR cues displayed for targeting and identification. The slower frame rate is therefore not considered detrimental to the VC application.

7.2 Software Reliability

The VC application has performed reliably in the different modes implemented. The hardest test on the robustness of the code was exhibiting the VC in the combined AFIT and NPS display during SIGGRAPH 93. During this six day convention the VC was operated for a total of 44 hours with 410 scheduled demonstrations given to individual viewers. During each day the VC came off the net an average of four times a day due to unexplained circumstances. Initial indications were from null pointer references towards Playerlist coordinates. The VC was routinely re-calibrated every hour to realign the Polhemus sensor with the graphics display since movement of the viewer's sitting position induced drift into the projected display.

7.3 Problems and Suggestions For Improvements

7.3.1 Floating Point Accuracy. The VC application loses fidelity due to the low resolution of floating point accuracy. This problem is not noticeable when implementing limited size flat earth terrain descriptions. When migrating to the round earth terrain description, the values for position information can soon reach in the millions of meters. Performer limits itself to single precision floating point computations in all of its functions. In addition, the Multigen modeller has difficulty in creating models that entail values and distances in the million meter range. This means that actual earth terrain descriptions must be offset from their actual physical position, and then translated to the correct position in the simulation. During Performer operations on large floating point numbers, some precision errors are encountered. These errors are noticeable in the round earth conversions necessary in transforming PDU positions to Performer reference frame positions. The evidence of this precision error is jittery movement of other round earth network entities in the simulation.

The composed matrix for rotations and translations for round to flat earth (C_{round}^{flat}) is put into the Performer tree root DCS of the terrain and is `pfFlatten`'ed into the terrain description so a transformation is not repeatedly done to the terrain. At the time of reading the terrain file, the transformation procedure is done once and the new rotated and translated orientation is the reference frame that all views of the terrain are based on. Since the computation is done once, there are no noticeable problems with significant digit rounding error visible with the terrain, even though an error actually does occur. Since the error is not recalculated each frame, the error difference remains the same from frame to frame and is not noticeable.

During experimentation with the terrain C_{round}^{flat} matrix, which was updated each frame, significant precision errors occurred (±80 meters) using a terrain model only $\frac{1^{th}}{10}$ the size of the earth. The accuracy of a single precision IEEE floating point number *provides seven significant* digits, and when the earth radius description is seven digits in meter length units, there is a loss of accuracy. This is most noticeable when interpreting and sending locations of network players and the VC's center of mass position. After the computations for each successive frame, the errors induced can lead to rough movements of the network models. As many computations as possible were accomplished using doubles to increase the significant digits to fifteen and allow the numbers to range from 1.7×10^{-308} to 1.7×10^{308}.

Since SGI workstations use single precision floats in the graphics pipeline calculations, to preserve accuracy until the final Coords-> values are determined, routines to mimic Performer's matrix operations can be built. This does not cause too much frame rate degradation since construction of most matrices is done once; this is in contrast to the application of the matrices on each net player's data every frame. Expanding the pre-composed matrices in the utility code to handle all combinations of matrices for each possible transformation would make every matrix available after initialization without having to repeat some matrix multiplications.

7.3.2 Mission Duration/Distance Travelled. The terrain is oriented once to a single local origin. The further the VC moves from the local origin, the more the curvature of the earth becomes a problem. The terrain is still a sphere, only positioned in flat local space. As the VC flys to the edge of the earth, the terrain continues to slope away. If the VC flys far enough, differences in flat earth straight and level flight, and tangent to the current round earth straight and level flight at the current location become noticeable. This difference is similar to the precession a pilot sees in some gyroscopic stabilized cockpit instruments (29:16)(6:4-7). To overcome this precession, a re-caging or fast slaving of the attitude gyros is accomplished to realign the instruments to the proper orientation.

For each 60 miles travelled from the origin, a 1° of attitude change is noticeable. Errors of 2° or less are barely noticed by the eye, and have little effect on pilot performance. When approaching 3° of difference the pilot will start compensating by either flying off the instruments in a cocked attitude that accounts for the precession, or caging the system to bring it back to alignment. If the instruments are not required for that phase of flying, the error usually is not noticed until the **Before Descent Checks** that are accomplished during the transition phase of the mission.

If 3° of error is acceptable to work with, this will allow 180 miles of operation from the local origin. If the origin is centered in the area of interest, then a usable terrain area of 360 × 360 miles is available. If there could be a way to have "intelligent" terrain know when the application is approaching the 180 mile limit, or whatever limit is specified, then a re-calibration to a new local origin could be accomplished (earthquake) and all errors removed. The ability to cause an earthquake unobtrusively in real time on the simulation terrain could be an area of further research.

Actual low level flight for distances in excess of those described is not very common because of fuel considerations. Most commonly a Hi-Low-Hi flight profile is used. At each transition into a Hi phase of the flight, the standard Performer horizon model could be maintained while the earthquake for the next low phase of the profile is accomplished. This is similar to what is seen in flight simulators with terrain model boards or computer generated terrain where a horizon reference line is substituted for the detailed terrain description after passing through about 5000 feet AGL. Since the mission planning materials specify exactly where and when each phase of the prospective flight profile occurs, the data for the earthquake computations is readily available (6:3-4).

7.3.3 Radar Enhancements.

7.3.3.1 Levels Of Damage. The radar has no capability to discern rubble or damage to an object in the simulation. The radar can only read PDU information and does not register actual changes in an object's structure. An area to increase the realism of the radar presentation would be to discern how much of a damaged object exists based on damage indications in PDUs and then scale the returned radar presentation according to the damage.

7.3.3.2 Terrain Returns. Terrain definition is lacking for the ground and navigation modes of the radar. There may be an economical way to superimpose network traffic data on a monochrome gray scale "moving map" of the terrain in the ground and navigation modes. There does not seem to be an easy way to cheaply get the accuracy of a "real" radar presentation.

7.3.3.3 Emitter PDUs and Radar Warning. Very little on radar emission PDU registration was accomplished since the priority for interpreting network traffic was focused on ensuring the accuracy of the entity state and weapons PDUs. The hooks for radar emitter PDUs are in the code, but this is an open area to investigate and should be accomplished since the radar is being used to locate entities. Emitter efforts would coincide with the development of radar warning sensors for the VC since similar data would need to be sent from one side (the radar), and interpreted by the warning or electronic countermeasures gear.

7.3.4 FLIR Enhancements. An interesting enhancement would be methods to take the attached FLIR view on a released weapon and fully simulate an EO guided weapon display. This would require allowing direct user inputs to control the action of a weapon system similar to an AGM-130. The VC has the ability to attach the FLIR view to an EO weapon. There is no capability at this time to actually direct the weapon with user inputs.

7.3.5 Initial VC Orientation. The current aerodynamic model does not seem to accept any other initial starting orientation other than zero degrees of heading, pitch, and roll. It has not been determined if this has to do with the initializing of the quaternions in the aerodynamics model, or some other intricacy of the model. The restricted orientation limits the starting position of the VC. The aerodynamics model also is limited in its range of parameters. The model assumes the VC is operating in the heart of the flight envelope (350 knots), and all coefficients are set for this flight environment. This causes the low speed characteristics of the VC to diverge from the actual aircraft aerodynamic characteristics. If a user is experimenting near the stall speed or attempting a take-off or landing, the VC is not as accurate as an aircraft in normal flight.

7.3.6 Arbitrating Damage In Distributed Simulations. We observed during SIGGRAPH 93 that when the NPS software and our VC were operating in a distributed environment, significant differences occurred in the amount of damage the simulations showed to the terrain. It appears that a more specific weapons effect description might be needed for the DIS standard. Each simulation site currently has an arbitrary capability to determine whether a static feature in the terrain is destroyed, damaged, or left alone when a weapon effect occurs. If one sight reports a destroyed bridge implementing a liberal application of a weapon blast or fragmentation effect using a large blast radius, and another sight only registers damage if that same weapon only destroys an object with a direct hit, then the terrain databases are no longer consistent.

7.3.7 VC Displays And Capabilities. The VC made great progress from VC 1.0 to VC 2.0. A list comparing capabilities between VC 1.0 and VC 2.0 and a possible future variant is described in table 4. Figures highlighting the different VC 2.0 display modes and capabilities are shown in figure 36 to figure 42. These are contrasted against the VC 1.0 instrument display of figure 35. Specific VC 2.0 operations, switchology, and weapon system interaction are described in the VC User's Manual of Appendix A.

7.4 Conclusions

The AFIT VC demonstrated the ease with which a low cost workstation can be configured to allow a user to interact with a distributed simulation environment. The weapons employment, detection, and pilot interactions are all open for expansion in the architecture of the ObjectSim and VC application programs. The malleability of the application was demonstrated in the different cockpit configurations and display devices used in the current VC. The VC shows the potential that low cost distributed flight simulations can attain with future research.

Capability	VC-1.0	VC-2.0	VC-X
Flight Fidelity			
Frames per Second	7	15	15+
Sound		X	X
Viewing Devices:			
CRT	X	X	X
Head-Mounted Display		X	X
miniDART		X	X
Stereo CRT		X	X
Head-Tracking		X	X
Network Compatibility			
SimNet			
Send/Receive Entity State	X		
Send/Receive Fire			
Send/Receive Detonation			
DIS			
Send/Receive Entity State		X	X
Send/Receive Fire		X	X
Send/Receive Detonation		X	X
Send/Receive Emission			X
Send/Receive Lasing			X
Send/Receive Collision			X
Coordinate Systems			
WGS-84		X	X
Flat Earth	X	X	X
Sensors			
RADAR			
Air-to-Air		X	X
Air-to-Ground		X	X
Navigation		X	X
FLIR			
Cue Mode		X	X
Track Mode		X	X
RADAR Warning Receiver			X
ECM/ECCM			X
Avionics			
Inertial Navigation System		X	X
Global Positioning System			X
Tactical Air Navigation			X
Instrumentation			
Simple Texture Map	X		
Polygon Descriptions based on F-15		X	X
Head-Up Display			
Flight Information		X	X
Targeting Information		X	X
CCIP		X	X
Fixed Gun Sight		X	X
Target Designation Box		X	X
Lead Computing Gun Sight			X
Weapons			
Gravity Bombs		X	X
LASER Guided Bombs		X	X
Electro-Optic Guided Bombs			X
Cannon		X	X
RADAR Guided Missile		X	X
IR Guided Missiles		X	X
Electro-Optic Guided Missiles			X
Weapon Systems Officer (Back-Seater)			X

Table 4. VC capabilities and future modifications.

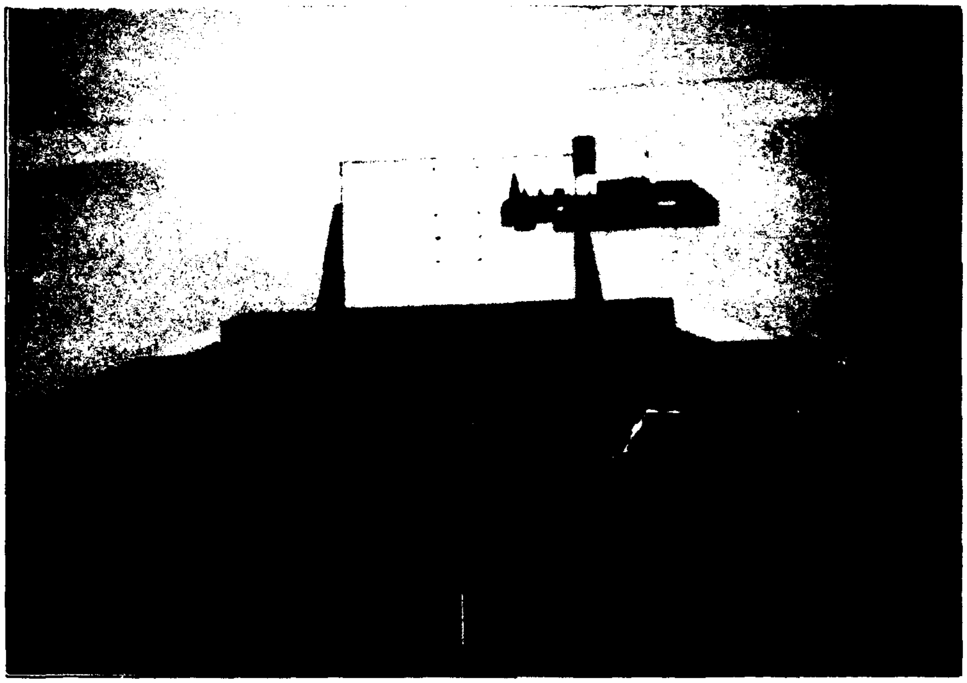


Figure 35. VC 1.0 Instrument Display

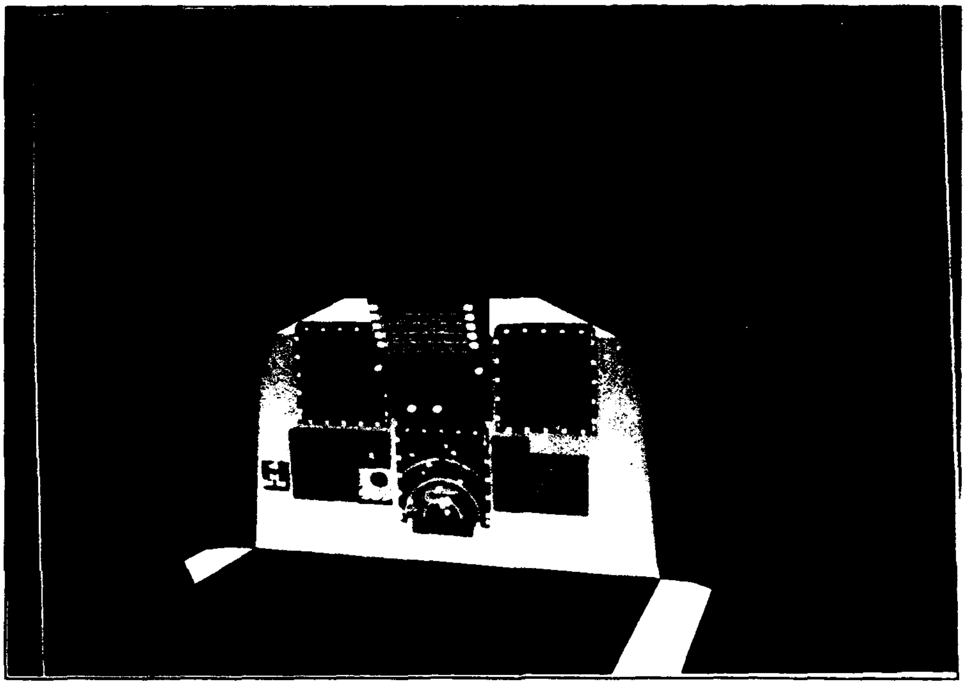


Figure 36. VC 2.0 Instrument Display Radar Ground Mode

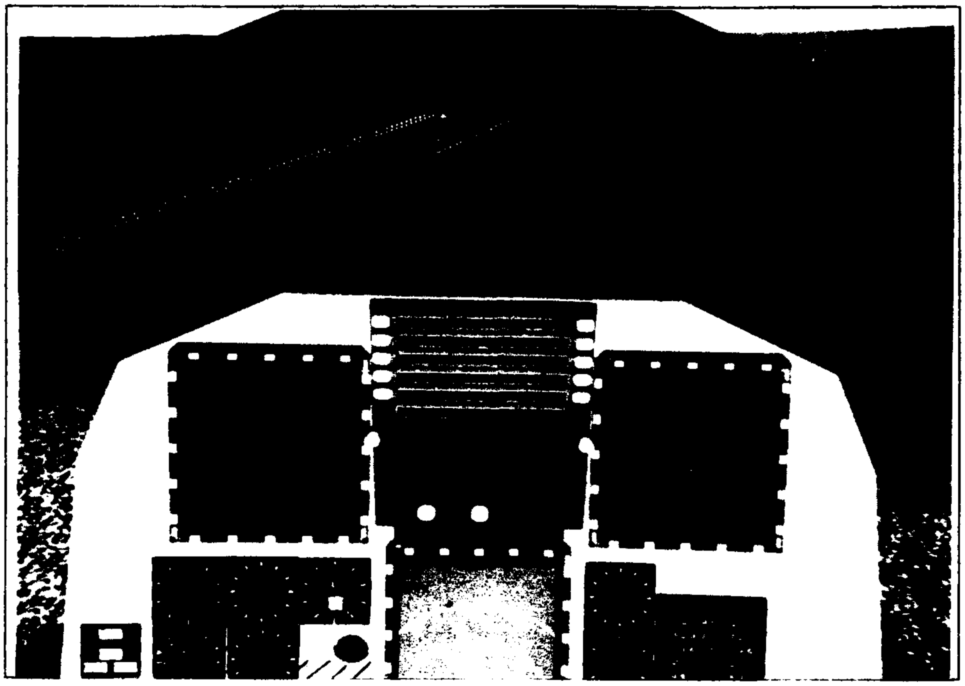


Figure 37. VC 2.0 Radar Navigation Mode

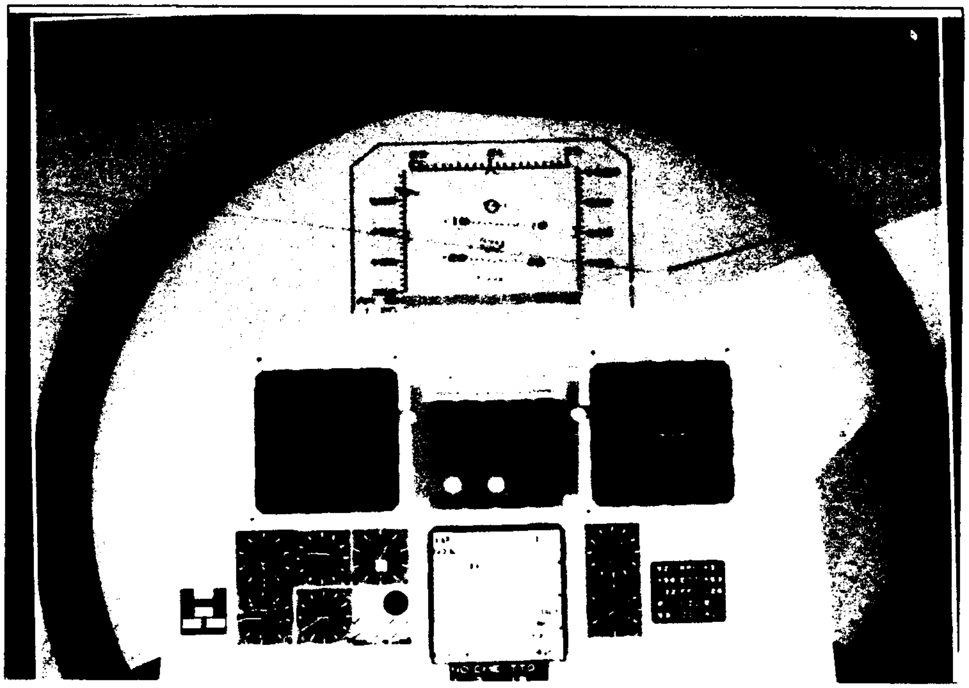


Figure 38. VC 2.0 Radar Air Mode With Target Designated

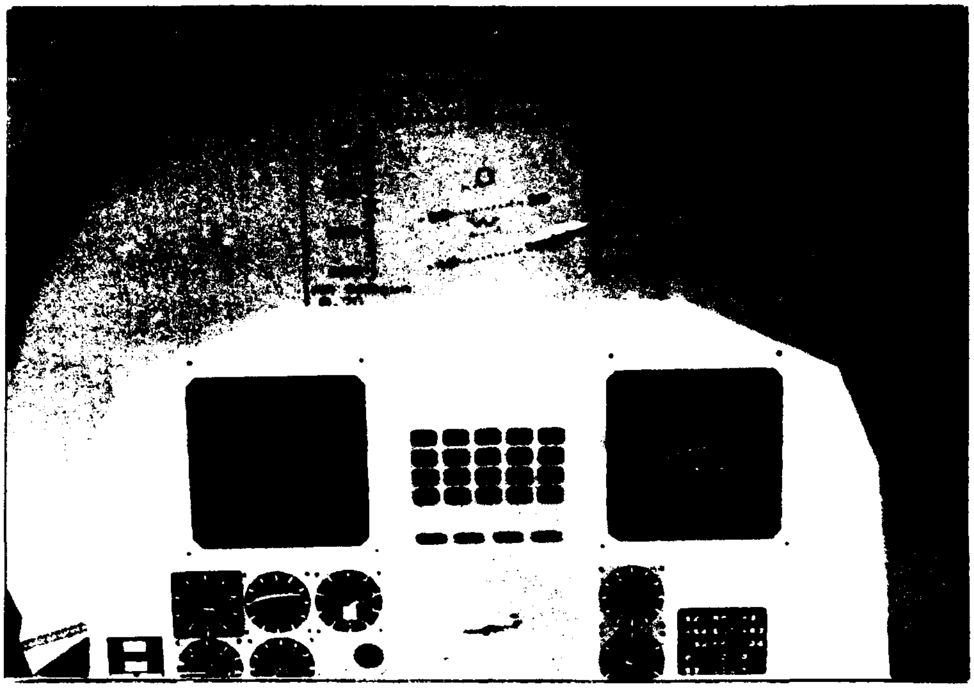


Figure 39. VC 2.0 FLIR Display With Air Target

Figure 40. VC 2.0 Radar Target With Radar Missile Launched And Tracking

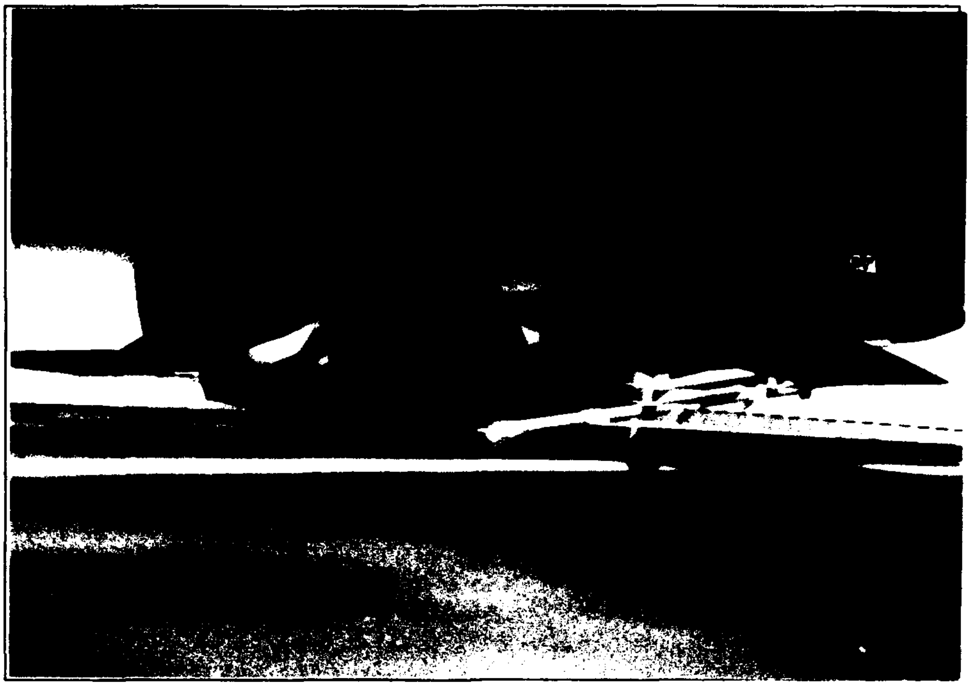


Figure 41. VC 2.0 Moving Parts Extended

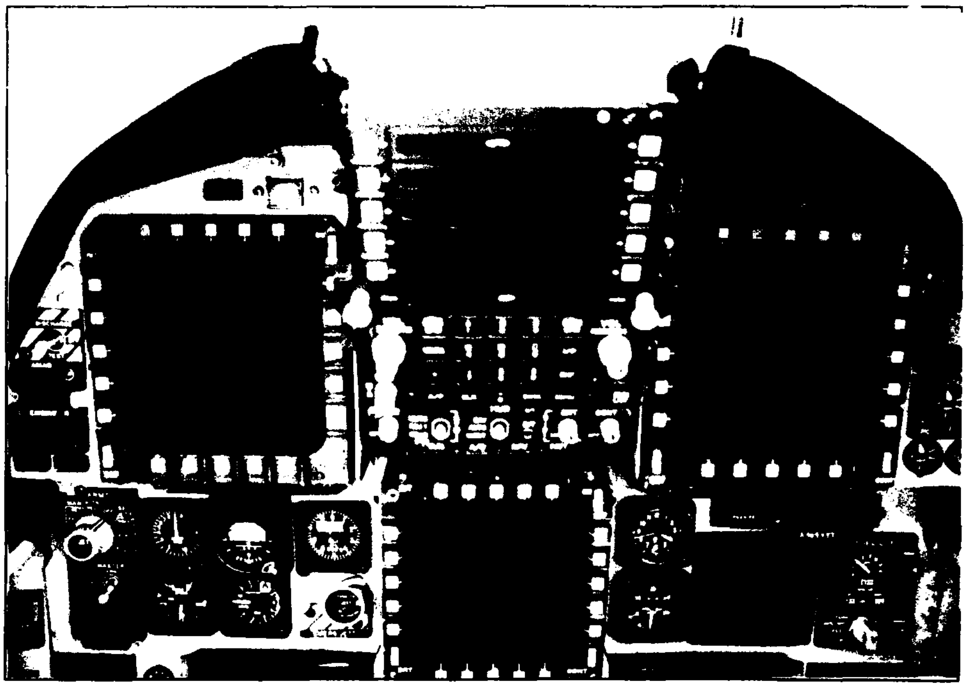


Figure 42. F-15E Forward Panel Front Cockpit

Appendix A. Virtual Cockpit User Guide

A.1 Introduction

The AFIT Virtual Cockpit (VC) is a research tool to investigate the application of low-cost, large-scale distributed interactive simulations for Air Force training and planning exercises. The VC is able to operate using the DIS protocols. The VC HOTAS has a basic switch configuration (Figure 43, Figure 44) with access to modified configurations depending on the HOTAS submode selected.

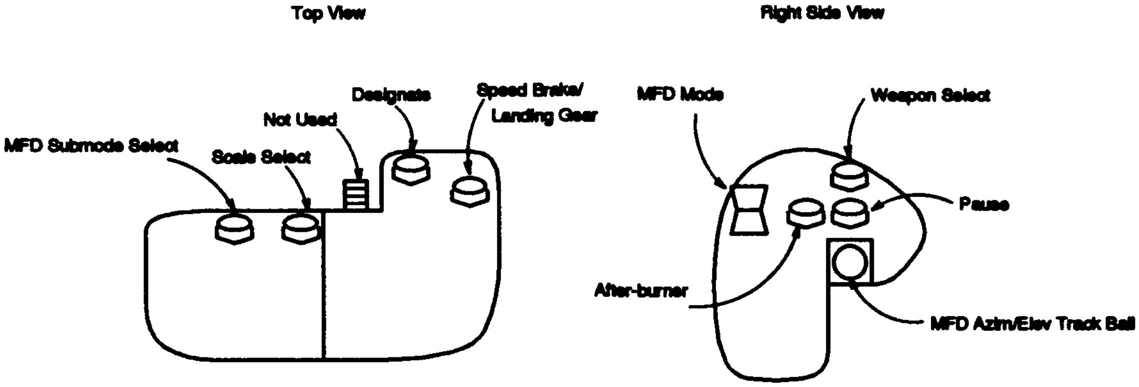


Figure 43. Basic Dual Throttle Switches

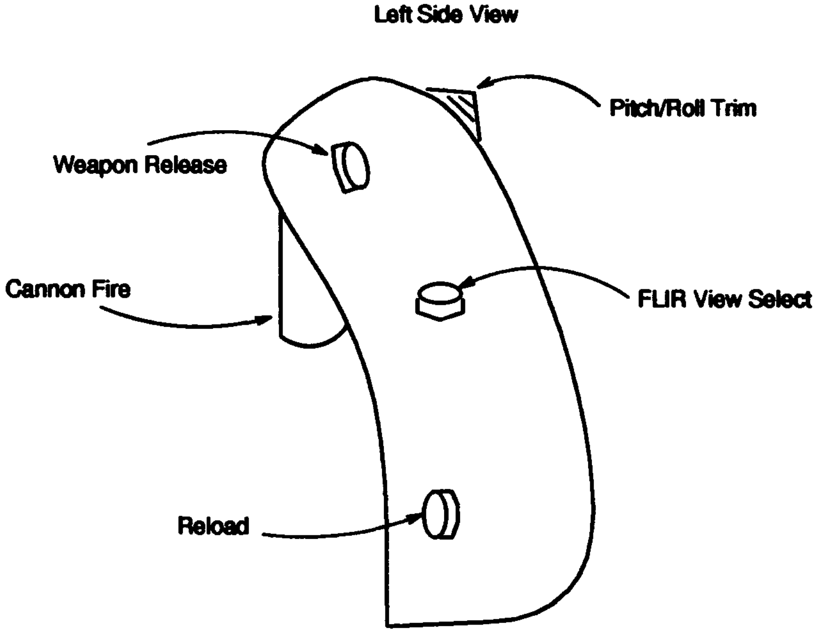


Figure 44. Basic Stick Switches

A.2 HOTAS Switches And Operation

A.2.1 Throttle Switches Radar Submode. The VC radar submode is entered by depressing the upper rocker of the multi-function display (MFD) Mode switch located on the right throttle. The switch description moves from left to right across the throttles (Figure 45).

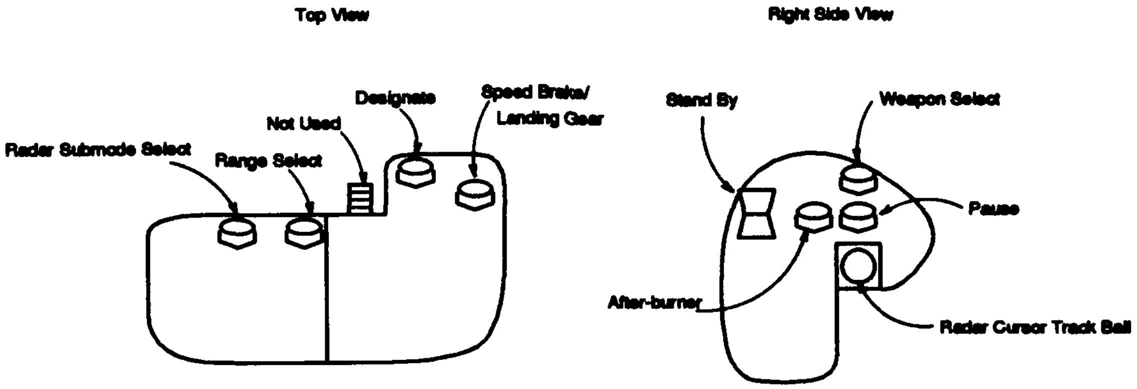


Figure 45. Radar Submode Throttle Switches

- Radar Submode Select: Toggles through ground, air, and navigation radar modes.
- Range Select: Toggles radar display through 160, 80, 40, 20, and 10 mile ranges.
- Not Used: Reserved for future modifications.
- Designate:
  - Air Mode: Selects the target that is under the radar cursor if no target is currently designated. If a target is designated, the current target is deselected when designate is depressed. If no target is under the cursor and a target is in the radar field of view, then the closest target is designated as the current radar target.
  - Ground Mode: Currently not implemented.

  - Navigation Mode: The next navigation route point of the navigation route point list is selected. This point is displayed if the point is visible in the radar.

- Speed Brake/Landing Gear: Triples aircraft drag to slow velocity while extending the speed-brake. Changes the speed brake indications in the cockpit. If aircraft velocity is under 250 knots the landing gear extends and changes to gear safe indications occur. If the speed brake and gear are extended, then activates retraction and changing of cockpit indications.

- Weapon Select: The available weapons for use in the cockpit are initialized in the weapons.config file. The selection of the weapons from the specified configuration is dependent on the radar mode selected.

  - Air Mode: Can toggle desired weapon selection through 20mm cannon, AIM-7 radar guided missile, and AIM-9 infra-red (IR) guided missile, AIM-120 radar guided missile, and WXA application experimental missile.

  - Ground Mode: Toggles desired weapon through 20 mm cannon; MK82, MK83, and MK84 conventional bombs; GBU12, GBU16, and GBU10 laser guided bombs; GBU15 and AGM65 optically guided munitions; and WXG application experimental bomb.

  - Navigation Mode: No effect.

- Pause: Stops aircraft movement while preserving current flight dynamics. Weapon systems remain functional.

- After-burner: Activates the after-burners when throttles are full forward and the button is momentarily depressed. After-burners de-selected by momentarily pulling the throttles back from the full forward position.

- MFD Mode:

- Radar Mode: Continuously updates targets in the radar field of view when the upper half of the toggle is depressed.
- Stand By: Ends radar emissions when toggle is centered, target positions on the radar display remain in their last displayed position.

- Radar Cursor Track Ball: Moves radar cursor vertically or horizontally on the display in relation to the trackball movement.

A.2.2 Throttle Switches FLIR Submode. The forward looking infra-red (FLIR) submode is entered by depressing the lower rocker of the MFD Mode switch located on the right throttle. The switch description moves from left to right across the throttles (Figure 46).

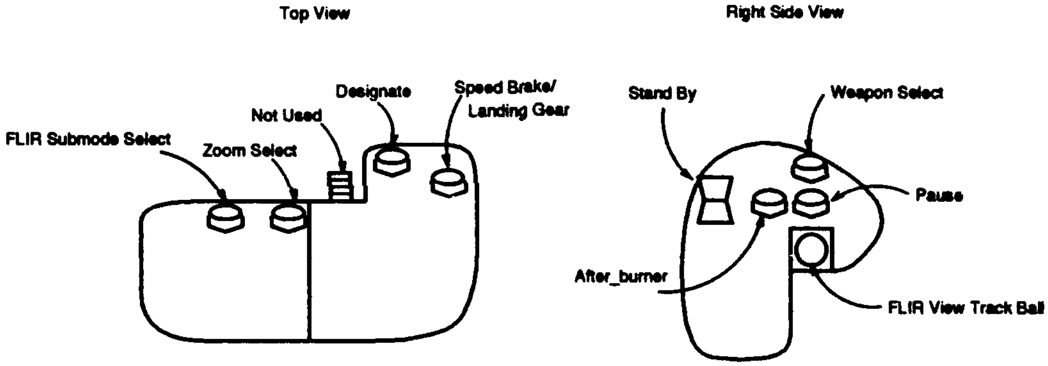


Figure 46. FLIR Submode Throttle Switches

- FLIR Submode Select: Toggles the FLIR between slaved view which is directed toward the selected target or navigation point, and manual track view which updates the view direction from the FLIR View Track Ball inputs.
- Zoom Select: Toggles FLIR display between an unmagnified view and a four times magnification view of the selected point.
- Not Used: Reserved for future modifications.

- Designate: Only active if selected from navigation radar mode. Will orient the FLIR view towards the next navigation route point in the navigation route point list.
- Speed Brake/Landing Gear: Same as Radar Submode.
- Weapon Select: Same as Radar Submode. The radar remains in the same mode that was selected prior to displaying the FLIR.
- Pause: Same as Radar Submode.
- After-burner: Same as Radar Submode.
- MFD Mode:
  - FLIR Mode: Overlays the MFD with a view oriented towards the point of interest when the lower half of the toggle is depressed.
  - Stand By: Same as Radar Submode, except the FLIR display remains oriented toward the desired view.
- FLIR View Track Ball: When in manual track mode, the FLIR view will move vertically or horizontally corresponding to the trackball movement.

A.2.3 - Stick Switches All Modes. The switch description moves from top to bottom on the control stick (Figure 44).

- Pitch/Roll Trim: Adjusts elevator/aileron deflection to maintain desired flight attitude.
- Weapon Release: Releases/Fires the bomb/missile which is selected on the head up display (HUD).
- Cannon Fire: Fires cannon rounds continuously when cannon selected on the HUD.
- FLIR View Select: Toggles the view between the FLIR pod field of view and a weapon field of view. If the selected weapon is an electro-optical guided

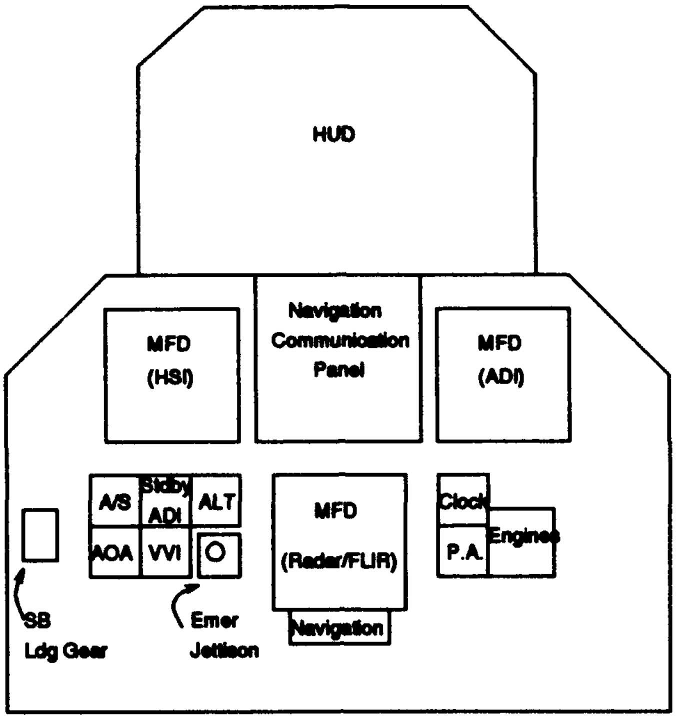


Figure 47. VC Named Layout

munition, the field of view will correspond to the munition warhead orientation. The view will propagate with the munition flight path.

- Reload: Restores original weapon load after released weapons have impacted and removes damage inflicted on the aircraft from a weapon event.

A.3 Running the Virtual Cockpit

A.3.1 Start Up. The rebuilding of the landscape from destroyed features assumes that a Polhemus head-tracker or a spaceball will be connected to Port 2. If the crippled Polhemus option (default) is used, the landscape can only be rebuilt by stopping the application and restarting. The hands on throttle and stick (HOTAS) must be installed in Port 3.

In the VC directory, entering `plane` will start the VC with the crippled Polhemus. Head look angles can be changed with the up/down, right/left arrows of the number pad. Head rotation clockwise/counter-clockwise are changed with keypad

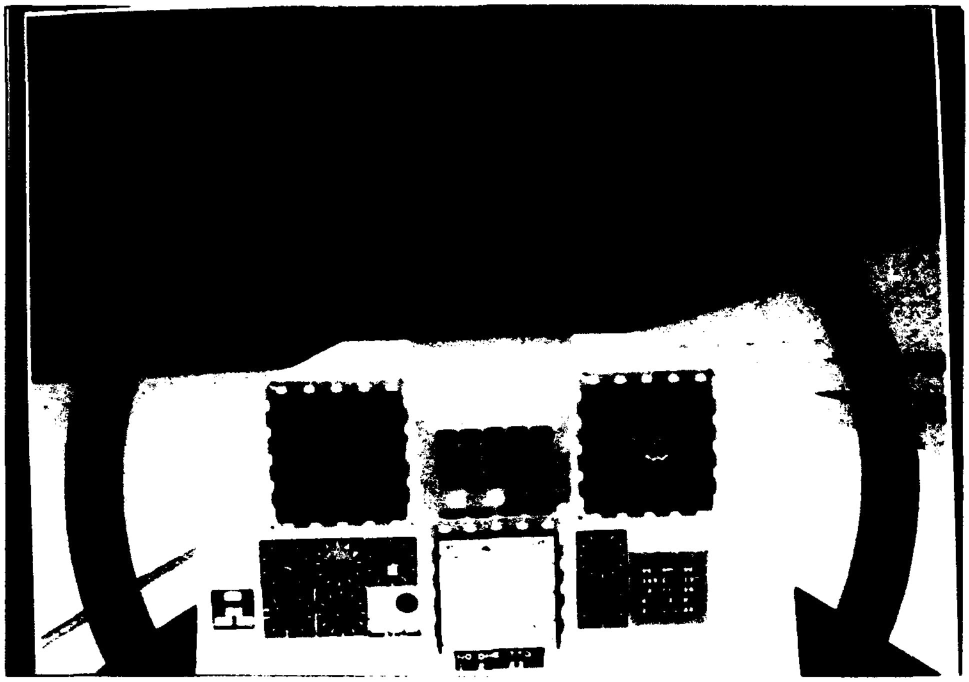


Figure 48. VC Layout

3/1 of the number pad. Up/down, left/right, forward/back head translations are accomplished with the u/d, l/r, and f/b keypads. Ensure the application process is killed after quitting the program. To reset the modified view to the original position, depress FIVE on the number pad. An overhead view of the airplane can be manipulated with the same keypads. To enter the overhead view, simultaneously depress the center and right mouse button. To return to the cockpit view, simultaneously depress the left and center mouse buttons.

`plane -p` will run the VC with the Polhemus head tracker and `plane -s` will run the VC with the space-ball. The space-ball option is available even if the space-ball is not hooked up. Using the space-ball option without actually connecting a space-ball will cause the view angles to freeze in the default position. Attempting to use the Polhemus option without a working Polhemus connected to the workstation will render the VC inoperative. If in either the Polhemus or space-ball modes, a flight timer is available to limit the user flight times to 1, 2, 3, 4, or 5 minute intervals. This is accomplished by depressing any of the keypads F1 through F5. To temporarily pause the VC press PAUSE. To quit the timer mode press END. To rebuild destroyed landscape press HOME.

Six initialization files are required to configure the VC.

1. `calibrate.sim` is required in the directory that the executable is run from (VC). This is the HOTAS specific calibration file that is generated from the command line script during start of the VC. If a new HOTAS is driving the VC in this directory, the calibration must be run to calculate valid HOTAS inputs.
2. `weapons.config` is required in the directory that the executable is run from (VC). This is a user built file specifying the weapon type and location for the chosen weapon load.
3. `F15.dat` is the aircraft-type specific aerodynamic coefficients for calibrating the aerodynamic model.
4. `init_sim` is the initial position file specifying start-up location.
5. `terrain_coords` is the description of the origin of the local reference frame in round earth coordinates and latitude/longitude conversion reference.
6. `nav_points.dat` is the route point description for loading INS coordinates and the bulls-eye reference.

A.3.2 MFD Description. The MFD operates in three modes: Radar, FLIR, and Standby. The Radar mode can show an air to air display, an air to ground display, and a navigation display. These modes affect the HUD symbols shown to the pilot as well as changing the information on the MFD. The most complete radar display is the air to air mode which will show line of sight targets within the specified radar range (Figure 49).

Some target information is available such as air target bearing and distance from the "Bullseye". The Bullseye is loaded as the last navigation point in the `nav_points.dat` file. All navigation point coordinates are entered as an x, y, z tuple in local flat earth terrain description. The target's heading, aspect angle, altitude in flight levels, velocity and closure velocity are displayed. The other radar mode descriptions were explained in the HOTAS switchology section.

The FLIR does not yet show an infra-red image. It is similar to a pitch and roll stabilized view you would see in an under-carriage infra-red (IR) pod. There are no gimbal limits, so the view can inadvertently be directed into the aircraft depending on target and aircraft orientation. The FLIR can be operated in three modes, track (TRK), cue (CUE), and weapon (WPN).

TRK orients the FLIR view toward the specific coordinates selected. In air to air radar mode the FLIR will look toward the coordinates of the selected air target. In air to ground radar mode the FLIR will look toward the coordinates of the selected ground target which is currently the most recently selected navigation point. In navigation radar mode the FLIR will look towards the coordinates of the selected navigation point.

CUE will allow the user to move the point of interest coordinates of the FLIR. This requires the look angle of the FLIR to intersect a valid portion of the terrain. When the cursor trackball is moved in CUE mode, the FLIR will move its view orientation to follow the trackball inputs and determine the intersection point with the terrain at that orientation. The CUE coordinates are saved until another trackball input is made and new cue coordinates are computed, or until TRK is selected. Reselecting TRK returns the coordinates to determine FLIR views.

WPN aligns the FLIR view with the longitudinal axis of the weapon selected. This view will simulate a fixed orientation electro-optical display for the weapon system. WPN view is only available if an appropriate EO weapon is selected for employment.

A.3.3 HUD Description. The HUD will show aircraft control and performance parameters as well as enable accurate weapon delivery (Figure 50). Aircraft airspeed is displayed along the left vertical side, heading along the top side, and altitude along the right vertical side. The bottom left corner will display your "G" state as well as prompt you which weapon mode you have entered from the HOTAS. Aircraft attitude is displayed with a waterline aircraft presentation superimposed on a rolling pitch ladder.

Air mode will show you gun rounds available or remaining missiles, depending on the weapon type selected. If gun is selected, a pipper calibrated to 1000 meters in ballistic trajectory from the aircraft is predicted. If in missile mode, a target designator box (TDB) surrounds the radar locked on target if the target is in the HUD field of view limits. If the target is out of limits, the TDB will give you steering directions to the target by displacing itself to the side of the HUD to which you should turn.

Ground mode will show you gun rounds available or remaining bombs. Gun sight is the same as air mode while the bomb fall line is projected from the flight path marker to the predicted bomb impact point giving the pilot a continuously computed impact point (CCIP).

Navigation mode gives steering information to the next navigation point through the deflection of the TDB. If over 90 degrees of turn from the selected navigation point, you will receive indications similar to back course steering information to the navigation point (get an old T-37 instructor to explain back course steering directions). Remaining mindful of the navigation point selected and your current position will keep you from turning the wrong direction.

A.3.4 Gotchas.

- If on startup the screen stays blank, or the HOTAS cannot be calibrated, reseat all HOTAS connections while ensuring the power connection is seated last.

You will need to kill the application and restart. If you are using the Polhemus or space-ball options, the Polhemus or space-ball may not be transmitting.

- You can create gigabytes of output log if you run for an extended time. To disable this "feature" make the link `/usr/tmp/send.log -> /dev/null` in `/usr/tmp`.
- If you can't change weapon modes make sure the MFD is not in stand by.
- Check Six.

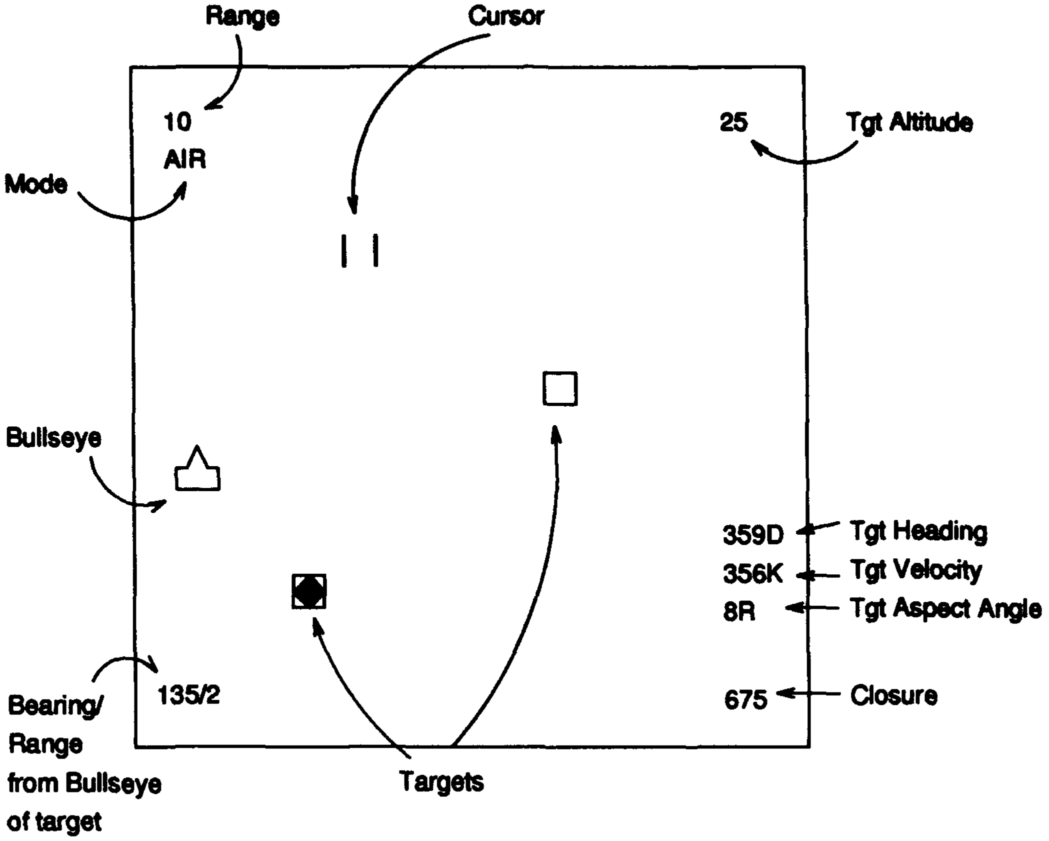


Figure 49. Sample MFD Display

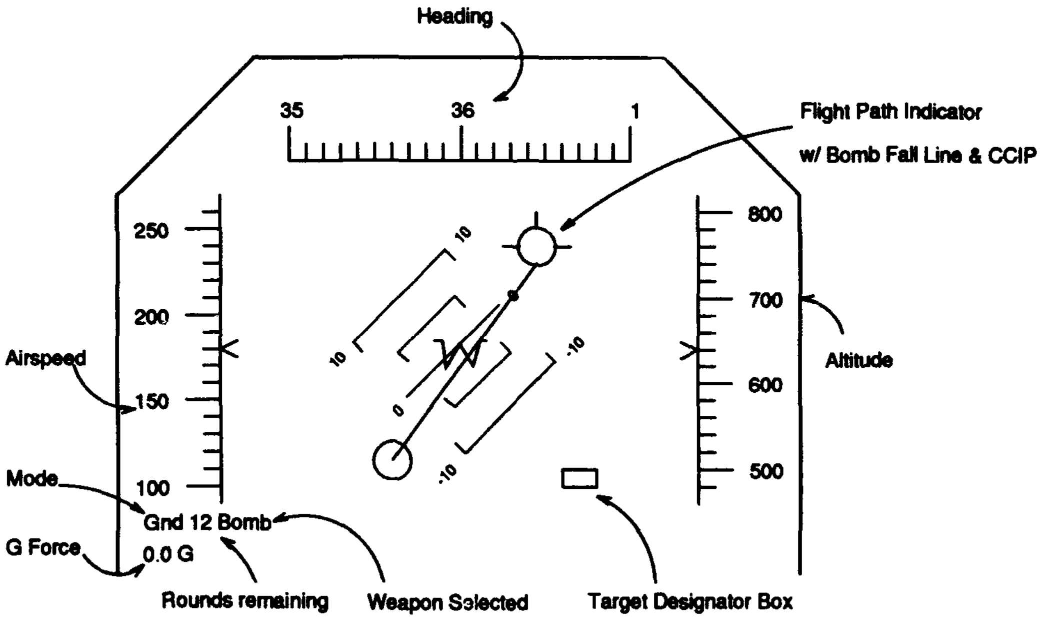


Figure 50. Sample HUD Displays

Appendix B. Round Earth Utilities

B.1 Header File

```
//  Class: Round_Earth_Utils
//
//  Purpose: Provide methods to transform a round earth orientation
// into a flat earth orientation, or flat earth to round earth.
// Transforms specific for the WRM 84 round earth definition.
// Flat earth can be at your discretion, virtual cockpit uses
// Y is North, X is East, and Z is Up (ENV).  Some methods for
// euler extraction borrowed heavily from Joe Cooke's
// aeromodel master's thesis at the NPS, and direction cosine
//         matrix (DCM) building from EENG 534 course notes Fall 92 at
//         AFIT.
//
//  Author: Matt Erichsen
//  Written in AT&T C++.
//  Last Modified: 28 Aug 93, limited testing complete (MNE).
//    29 Aug 93, Added routines to get satellite coordinates
//    29 Aug 93, Added routines to flip in Performer reference
//       frames, or any frames desired
//    29 Aug 93, Added routine to return MSL altitude for flat
//             earth coordinate description
//    30 Aug 93, Added routine to create segment from flat earth
//                            position to flat earth center.
//                   28 Oct 93, Added +-180 heading DIS standard check
//                            and latitude longitude conversion//
//  Copyright: Air Force Institute of Technology, 1993.
//  Released into the public domain.
//  Suggestions for improvement:  1. Provide a way to keep the double
//      precision needed for large earth
//      calculations.
//
//////////////////////////////////////////////////////////////////////////

#ifndef __REUTILS__
#define __REUTILS__

#include <stream.h>
#include <stdio.h>
#include <math.h>
#include <pf.h>

#define RAD_TO_DEGR  57.295779
```

```
#define DEGR_TO_RAD  0.0174533
#define SECONDS_PER_DAY 86400
#define TRUE 1
#define FALSE 0

class Round_Earth_Utils {

private:

  pfMatrix  FTR_2, RTF_2, ftr_trans_2, rtf_trans_2;
  void      round_flat_xforms_2(double x, double y, double z,
                        pfMatrix rtf, pfMatrix ftr);
  void      flat_pos_to_rnd_2(double& x, double& y, double& z);

public:

  Round_Earth_Utils();

  ///////////////////////////////////////////////////////////////////////////
  //  The correction matrices describe the rotations necessary to take an aero
  //  coordinate system at 0, 0, 0, degrees rotation, and make the aero body
  //  look like it is oriented North with 0 degrees pitch and roll.
  ///////////////////////////////////////////////////////////////////////////

  pfMatrix  FTR, RTF, ftr_trans, rtf_trans, rtf_twist, ftr_twist;
  pfVec3    flat_earth_center;
  double    Spsi, Stheta, Sphi, local_origin_radius;
  int       flat_earth;

  ///////////////////////////////////////////////////////////////////////////
  //  Enter position in WRM 84 round earth postion and will return the
  //  transformation matrices required to make the round patch a local level
  //  (rtf) and transform the local level information to round earth
  //  WRM 84 (ftr).
  ///////////////////////////////////////////////////////////////////////////

  void    round_flat_xforms(double x, double y, double z,
                    pfMatrix rtf, pfMatrix ftr);

  ///////////////////////////////////////////////////////////////////////////
  //  Given a direction cosine matrix (euler_dcm), return the euler angles
  //  describing the object's orientation in degrees, psi is the pitchfork,
//  theta is the circle with horizontal arc, phi is the circle with the
//  vertical line.
//////////////////////////////////////////////////////////////////////////////

void    euler_angles_from_matrix(double& psi, double& theta, double& phi,
                                 pfMatrix euler_dcm);
void    eeng_euler_angles_from_matrix(double& psi, double& theta,
                                      double& phi, pfMatrix euler_dcm);

//////////////////////////////////////////////////////////////////////////////
//  Given a set of euler angles, return the direction cosine matrix
//  describing the orientation.  This matrix is in standard aero model
//  format.
//////////////////////////////////////////////////////////////////////////////

void    matrix_from_euler_angles(double psi, double theta, double phi,
                                 pfMatrix dcm);
void    singularity_matrix_from_euler_angles(double psi, double theta,
     double phi, pfMatrix dcm);

//////////////////////////////////////////////////////////////////////////////
//  Given a set of euler angles, return the Performer h, p, r to orient the
//  model descriptions of the object in degrees
//////////////////////////////////////////////////////////////////////////////

void    euler_to_pfmr_hpr(pfVec3 hpr, double psi, double theta, double phi);

//////////////////////////////////////////////////////////////////////////////
//  Given a set of Performer h, p, r, return the euler angles to orient the
//  model descriptions in degrees
//////////////////////////////////////////////////////////////////////////////

void    pfmr_to_euler_hpr(pfVec3 hpr, double& psi, double& theta,
  double& phi);

//////////////////////////////////////////////////////////////////////////////
//  Given a set of euler angles in round earth description, transform them to
//  a flat earth euler angle representation, all angles in degrees. Requires
//  round_flat_xforms to have been done to initialize the rotation matrices.
//  TESTED AND WORKS CLOSE ENOUGH FOR NOW, needs work on singularity?
//////////////////////////////////////////////////////////////////////////////
```

```
void    rnd_euler_to_flat(double& psi, double& theta, double& phi);
void    singularity_rnd_euler_to_flat();

////////////////////////////////////////////////////////////////////////////
//  Given a set of euler angles in flat earth description, transform them to
//  a round earth euler angle representation, all angles in degrees. Requires
//  round_flat_xforms to have been done to initialize the rotation matrices.
//  TESTED AND WORKS CLOSE ENOUGH FOR NOW, needs work on singularity?
////////////////////////////////////////////////////////////////////////////

void    flat_euler_to_rnd(double& psi, double& theta, double& phi);
void    singularity_flat_euler_to_rnd();

////////////////////////////////////////////////////////////////////////////
//  Given a position in round earth description, transform that position to
//  a flat earth position, all distances in meters.
////////////////////////////////////////////////////////////////////////////

void    rnd_pos_to_flat(double& x, double& y, double& z);

////////////////////////////////////////////////////////////////////////////
//  Given a position in flat earth description, transform that position to
//  a round earth position, all distances in meters.
////////////////////////////////////////////////////////////////////////////

void    flat_pos_to_rnd(double& x, double& y, double& z);

/////////////////////////////////////////////////////////////////////////////
//  Given a vector in round earth description, transform that vector to
//  a flat earth vector.
/////////////////////////////////////////////////////////////////////////////

void    rnd_vec_to_flat(double& x, double& y, double& z);

/////////////////////////////////////////////////////////////////////////////
//  Given a vector in flat earth description, transform that vector to
//  a round earth vector.
/////////////////////////////////////////////////////////////////////////////

void    flat_vec_to_rnd(double& x, double& y, double& z);

////////////////////////////////////////////////////////////////////////////
```

```
// Given a position in flat earth description, returns what the altitude
// MSL is in meters for that coordinate description.
/////////////////////////////////////////////////////////////////////////////

double  alt_msl(double& x, double& y, double& z);

/////////////////////////////////////////////////////////////////////////////
// Given a satellite position in an earth centered inertial frame (ECI),
// and the seconds elapsed in the current day, transform that position to
// an earth centered earth fixed frame (ECEF), all distances in meters.
/////////////////////////////////////////////////////////////////////////////

void    ECI_to_ECEF(double& x, double& y, double& z, int seconds);

/////////////////////////////////////////////////////////////////////////////
// Given a satellite position in an earth centered earth fixed frame (ECEF),
// and the seconds elapsed in the current day, transform that position to
// an earth centered inertial frame (ECI), all distances in meters.
/////////////////////////////////////////////////////////////////////////////

void    ECEF_to_ECI(double& x, double& y, double& z, int seconds);

/////////////////////////////////////////////////////////////////////////////
// Creates a rotation matrix that can modify an euler angle description so
// the eulers are accurate in the ECEF frame.  As an example, the virtual
// cockpit aero model operates in an NED frame while the eulers have to be
// in an ENV frame to start aligned with the ECEF, this requires a rotation
// of 90 degrees around the Z axis (z_rot = 90.0) and 180 degrees around the
// X axis (x_rot = 180.0).  This routine MUST be called each time after
// round_flat_xforms if you want a modification, or else the rotation matrix
// will be the identity matrix. Rotation order will be z azis, y axis, then
// x axis.
/////////////////////////////////////////////////////////////////////////////

void    reference_frame_mod_ftr(double z_rot, double y_rot, double x_rot);

/////////////////////////////////////////////////////////////////////////////
// Creates a rotation matrix that can modify an euler angle description so
// the eulers are accurate in the ECEF frame. This routine MUST be called
// each time after round_flat_xforms if you want a modification, or else the
// rotation matrix will be the identity matrix.
/////////////////////////////////////////////////////////////////////////////
```

```
  void    reference_frame_mod_rtf(double z_rot, double y_rot, double x_rot);

  ////////////////////////////////////////////////////////////////////////////////
  //  Given a position in flat earth description, returns a segment that is
  //  directed from your flat earth position to the ''flat'' center (core)
  //  of the earth.
  ////////////////////////////////////////////////////////////////////////////////

  pfSeg  core_segment(double x, double y, double z);


  ////////////////////////////////////////////////////////////////////////////////
  //  Given a position in flat earth description, returns the latitude and
  //  longitude that corresponds to the position.  Degrees returned with
  //  decimal notation for partial degree, not minutes and seconds
  ////////////////////////////////////////////////////////////////////////////////
  void  lat_long_from_xyz(double x, double y, double z,
                          double orig_x, double orig_y, double orig_z,
  float& Latitude,  float& Longitude);


};


#endif
```

B.2 Body File

```
#include "round_earth_utils.h"


Round_Earth_Utils::Round_Earth_Utils()
{

}

///////////////////////////////////////////////////////////////////////////
//
///////////////////////////////////////////////////////////////////////////
void Round_Earth_Utils::round_flat_xforms(double x, double y, double z,
  pfMatrix rtf, pfMatrix ftr)
{

  pfVec3    cur_pos;
  pfVec2    temp;
  double    theta, beta, sintheta, sinbeta, xyzplus2, xyplus2;
  pfMatrix  x_rot, z_rot, pfz_rot, pfy_rot, pfx_rot, trans_rtf, trans_ftr;

  if (x == 0.0f && y == 0.0f && z == 0.0f)
  {
   flat_earth = TRUE;
   pfMakeIdentMat(ftr_twist);
   pfMakeIdentMat(rtf_twist);
   pfMakeIdentMat(FTR);
   pfMakeIdentMat(RTF);
   pfMakeIdentMat(ftr);
   pfMakeIdentMat(rtf);
   pfMakeIdentMat(ftr_trans);
   pfMakeIdentMat(rtf_trans);
  }
  else
  {


//  cout << "rfx called"<<endl;
  flat_earth = FALSE;

  pfMakeIdentMat(ftr_twist);
pfMakeIdentMat(rtf_twist);

PFSET_VEC3(cur_pos, x, y, z);

PFSET_VEC2(temp, x, y);

xyzplus2 = PFLENGTH_VEC3(cur_pos);

xyplus2  = PFLENGTH_VEC2(temp);

pfSetVec3(flat_earth_center, 0.0, 0.0, -xyzplus2);

local_origin_radius = xyzplus2;

// Convert HPR of attached round earth player into Performer space

sintheta = cur_pos[PF_X]/xyplus2;

sinbeta  = xyplus2/xyzplus2;

theta    = pfArcSin(sintheta);  //angle needed for rotation around Z axis

beta     = pfArcSin(sinbeta);   //angle needed for rotation around X axis

// determine which quadrant angle is in and then determine by right hand rule
// how much of a rotation around the desired axis is needed

if ( x > 0.0f )                    // X-Y plane rotation around Z
{
  if ( y > 0.0f )       // quadrant 1
    theta = -(180.0f - theta);
  else    // quadrant 4
    theta = -theta;
}
else
{
  if ( y > 0.0f )     // quadrant 2
    theta = 180.0f + theta;
  else    // quadrant 3
    theta = -theta;
}
```

```
  if ( z > 0.0f )        // Y-Z plane rotation around X
    beta = -beta;                                    // quadrant 2
  else
    beta = -(180.0f - beta);   // quadrant 3


  pfMakeRotMat(z_rot, theta, 0.0f, 0.0f, 1.0f);  //heading

  pfMakeRotMat(x_rot, beta,  1.0f, 0.0f, 0.0f);  //pitch

  pfMultMat(rtf, z_rot, x_rot);  //this is the required order of mult for the terr

  //make transpose before inserting translations so proper translation occurs

  pfTransposeMat(ftr, rtf);

  pfCopyMat(FTR, ftr);

  pfCopyMat(RTF, rtf);

  rtf[3][2] = -xyzplus2;

  ftr[3][0] = x;
  ftr[3][1] = y;
  ftr[3][2] = z;


  pfCopyMat(ftr_trans, ftr);

  pfCopyMat(rtf_trans, rtf);
  }

}

//////////////////////////////////////////////////////////////////////////////
// This will extract from an eeng534 matrix, not needed at this point
//////////////////////////////////////////////////////////////////////////////
void Round_Earth_Utils::eeng_euler_angles_from_matrix(double& psi, double& theta,
                               double& phi, pfMatrix euler_dcm)
{
    double sptch, cptch, shdg, chdg, sroll, croll;
    sptch = -euler_dcm[2][0];     /*sinpitch*/
    cptch = fsqrt(1.0-(euler_dcm[2][0])*(euler_dcm[2][0])); /*cospitch*/
    if (cptch == 0.0)              /*if at +/- 90deg*/    {
    {
      shdg  = 0.0; /*sinyaw */
      chdg  = 1.0; /*cosyaw */
      sroll = euler_dcm[0][1]; /*sinroll*/
      croll = euler_dcm[1][1]; /*cosroll*/
    }
    else
    {
      shdg  = euler_dcm[1][0]/(cptch); /*sinyaw  */
      chdg  = euler_dcm[0][0]/(cptch); /*cosyaw  */
      sroll = euler_dcm[2][1]/(cptch); /*sinroll */
      croll = euler_dcm[2][2]/(cptch); /*cosroll */
    }

    theta = (fasin((sptch)) * RAD_TO_DEGR);

    phi   = (fasin((sroll)) * RAD_TO_DEGR);

    if (croll < 0.0) {
      if (sroll < 0.0) phi = (-180.0 - phi);
      else             phi = (180.0 - phi);
    }

    psi   =  (fasin((shdg)) * RAD_TO_DEGR);

    if (shdg < 0.0)
    {
     // for these two lines, psi < 0
     if (chdg < 0.0) psi = 180.0 - psi;
     else               psi = 360.0 + psi;
  }
  else
     if (chdg < 0.0) psi = 180.0 - psi;

}

/////////////////////////////////////////////////////////////////////////
// performer will create a matrix that needs this extraction
// [0][0] [0][1] [0][2] [0][3]
// [1][0] [1][1] [1][2] [1][3]
// [2][0] [2][1] [2][2] [2][3]
// [3][0] [3][1] [3][2] [3][3]
//////////////////////////////////////////////////////////////////////
void Round_Earth_Utils::euler_angles_from_matrix(double& psi, double& theta,
                              double& phi, pfMatrix euler_dcm)
{

    double sptch, cptch, shdg, chdg, sroll, croll;
    sptch = -euler_dcm[0][2];     /*sinpitch*/
    cptch = fsqrt(1.0-(euler_dcm[0][2])*(euler_dcm[0][2])); /*cospitch*/
    if (cptch == 0.0)              /*if +/- 90deg*/
    {
      shdg  = 0.0; /*sinyaw */
      chdg  = 1.0; /*cosyaw */
      sroll = euler_dcm[1][0]; /*sinroll*/
      croll = euler_dcm[1][1]; /*cosroll*/
    }
    else
    {
      shdg  = euler_dcm[0][1]/(cptch); /*sinyaw  */
      chdg  = euler_dcm[0][0]/(cptch); /*cosyaw  */
      sroll = euler_dcm[1][2]/(cptch); /*sinroll */
      croll = euler_dcm[2][2]/(cptch); /*cosroll */
    }

    theta = (fasin((sptch)) * RAD_TO_DEGR);

    phi   = (fasin((sroll)) * RAD_TO_DEGR);

    if (croll < 0.0) {
      if (sroll < 0.0) phi = (-180.0 - phi);
      else             phi = (180.0 - phi);
    }

    psi   =  (fasin((shdg)) * RAD_TO_DEGR);

    if (shdg < 0.0)
    {
      // for these two lines, psi < 0
     if (chdg < 0.0) psi = 180.0 - psi;
     else               psi = 360.0 + psi;
  }
  else
     if (chdg < 0.0) psi = 180.0 - psi;

}

/////////////////////////////////////////////////////////////////////////
//     This makes the EENG534 DCM representation
```

$$C_b^n = \begin{bmatrix} \cos\psi\cos\theta & \cos\psi\sin\theta\sin\phi - \sin\psi\cos\phi & \cos\psi\sin\theta\cos\phi + \sin\psi\sin\phi \\ \sin\psi\cos\theta & \sin\psi\sin\theta\sin\phi + \cos\psi\cos\phi & \sin\psi\sin\theta\cos\phi - \cos\psi\sin\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{bmatrix} = c_{ij} \tag{18}$$

```
/////////////////////////////////////////////////////////////////////////
void Round_Earth_Utils::singularity_matrix_from_euler_angles(double psi,
 double theta, double phi, pfMatrix dcm)
{
  double  Phi, Theta, Psi;
  double  cosphi, costheta, cospsi, sinphi, sintheta, sinpsi;

  Phi = phi*DEGR_TO_RAD; Theta = theta*DEGR_TO_RAD; Psi = psi*DEGR_TO_RAD;
  cosphi = cos(Phi);  costheta = cos(Theta);  cospsi = cos(Psi);
  sinphi = sin(Phi);  sintheta = sin(Theta);  sinpsi = sin(Psi);

  dcm[0][0] = cospsi*costheta;
  dcm[1][0] = sinpsi*costheta;
  dcm[2][0] = -sintheta;
  dcm[3][0] = 0.0f;

  dcm[0][1] = cospsi*sintheta*sinphi - sinpsi*cosphi;
  dcm[1][1] = sinpsi*sintheta*sinphi + cospsi*cosphi;
  dcm[2][1] = costheta*sinphi;
  dcm[3][1] = 0.0f;

  dcm[0][2] = cospsi*sintheta*cosphi + sinpsi*sinphi;
  dcm[1][2] = sinpsi*sintheta*cosphi - cospsi*sinphi;
  dcm[2][2] = costheta*cosphi;
  dcm[3][2] = 0.0f;

  dcm[0][3] = 0.0f;
  dcm[1][3] = 0.0f;
  dcm[2][3] = 0.0f;
  dcm[3][3] = 1.0f;
```

```
}

////////////////////////////////////////////////////////////////////////////////
//  This makes the performer/sgi oriented matrix that maintains
//  euler consistency
////////////////////////////////////////////////////////////////////////////////
void Round_Earth_Utils::matrix_from_euler_angles(double psi, double theta,
         double phi, pfMatrix dcm)
{
  double  Phi, Theta, Psi;
  double  cosphi, costheta, cospsi, sinphi, sintheta, sinpsi;

  Phi = phi*DEGR_TO_RAD; Theta = theta*DEGR_TO_RAD; Psi = psi*DEGR_TO_RAD;
  cosphi = cos(Phi);  costheta = cos(Theta);  cospsi = cos(Psi);
  sinphi = sin(Phi);  sintheta = sin(Theta);  sinpsi = sin(Psi);

  dcm[0][0] = cospsi*costheta;
  dcm[0][1] = sinpsi*costheta;
  dcm[0][2] = -sintheta;
  dcm[0][3] = 0.0f;

  dcm[1][0] = cospsi*sintheta*sinphi - sinpsi*cosphi;
  dcm[1][1] = sinpsi*sintheta*sinphi + cospsi*cosphi;
  dcm[1][2] = costheta*sinphi;
  dcm[1][3] = 0.0f;

  dcm[2][0] = cospsi*sintheta*cosphi + sinpsi*sinphi;
  dcm[2][1] = sinpsi*sintheta*cosphi - cospsi*sinphi;
  dcm[2][2] = costheta*cosphi;
  dcm[2][3] = 0.0f;

  dcm[3][0] = 0.0f;
  dcm[3][1] = 0.0f;
  dcm[3][2] = 0.0f;
  dcm[3][3] = 1.0f;

}

////////////////////////////////////////////////////////////////////////////////
//
////////////////////////////////////////////////////////////////////////////////
void Round_Earth_Utils::euler_to_pfmr_hpr(pfVec3 hpr, double psi,
                                            double theta, double phi)
{

  pfSetVec3(hpr, -psi, theta, phi);

}

////////////////////////////////////////////////////////////////////////////////
//
////////////////////////////////////////////////////////////////////////////////
void Round_Earth_Utils::pfmr_to_euler_hpr(pfVec3 hpr, double& psi,
  double& theta, double& phi)
{

  psi   = -hpr[PF_H];
  theta =  hpr[PF_P];
  phi   =  hpr[PF_R];

}

////////////////////////////////////////////////////////////////////////////////
//
////////////////////////////////////////////////////////////////////////////////
void Round_Earth_Utils::rnd_euler_to_flat(double& psi, double& theta,
                                          double& phi)
{
  pfMatrix edcm, x_rot, z_rot, y_rot, rnd_euler;

  matrix_from_euler_angles(psi, theta, phi, rnd_euler);
  pfMultMat(rnd_euler, rnd_euler, RTF);
  pfMultMat(edcm, rnd_euler, rtf_twist);
  euler_angles_from_matrix(psi, theta, phi, edcm);
  if (psi < -180.0)
    psi = 360.0 + psi;
  if (psi > 180.0)
    psi = psi - 360.0;

}

////////////////////////////////////////////////////////////////////////////////
//
////////////////////////////////////////////////////////////////////////////////
```

```
void Round_Earth_Utils::flat_euler_to_rnd(double& psi, double& theta,
                                          double& phi)
{
  pfMatrix edcm, x_rot, z_rot, y_rot, flat_euler;

  Spsi = psi;
  Stheta = theta;
  Sphi = phi;

  matrix_from_euler_angles(psi, theta, phi, flat_euler);
  pfMultMat(flat_euler, flat_euler, ftr_twist);
  pfMultMat(edcm, flat_euler, FTR);
  euler_angles_from_matrix(psi, theta, phi, edcm);
  if (psi < -180.0)
    psi = 360.0 + psi;
  if (psi > 180.0)
    psi = psi - 360.0;

}

////////////////////////////////////////////////////////////////////////
//  This was an experiment and is left if further problems occur
////////////////////////////////////////////////////////////////////////
void Round_Earth_Utils::singularity_flat_euler_to_rnd()
{
  pfMatrix edcm, singularity_matrix;

  singularity_matrix_from_euler_angles(Spsi, Stheta, Sphi, edcm);
  pfMultMat(singularity_matrix, FTR, edcm);
  eeng_euler_angles_from_matrix(Spsi, Stheta, Sphi, singularity_matrix);
}

////////////////////////////////////////////////////////////////////////////
//
////////////////////////////////////////////////////////////////////////////
void Round_Earth_Utils::rnd_pos_to_flat(double& x, double& y, double& z)
{
  pfVec3 pos;

  pfSetVec3(pos, x, y, z);
  pfXformPt3(pos, pos, rtf_trans);
  x = pos[PF_X]; y = pos[PF_Y]; z = pos[PF_Z];
}

/////////////////////////////////////////////////////////////////////////////
//
/////////////////////////////////////////////////////////////////////////////
void Round_Earth_Utils::flat_pos_to_rnd(double& x, double& y, double& z)
{
  pfVec3 pos;

  pfSetVec3(pos, x, y, z);
  pfXformPt3(pos, pos, ftr_trans);
  x = pos[PF_X]; y = pos[PF_Y]; z = pos[PF_Z];

}

/////////////////////////////////////////////////////////////////////////////
//
/////////////////////////////////////////////////////////////////////////////
void Round_Earth_Utils::rnd_vec_to_flat(double& x, double& y, double& z)
{
  pfVec3 vec;

  pfSetVec3(vec, x, y, z);
  pfXformPt3(vec, vec, RTF);
  x = vec[PF_X]; y = vec[PF_Y]; z = vec[PF_Z];

}

/////////////////////////////////////////////////////////////////////////////
//
/////////////////////////////////////////////////////////////////////////////
void Round_Earth_Utils::flat_vec_to_rnd(double& x, double& y, double& z)
{
  pfVec3 vec;

  pfSetVec3(vec, x, y, z);
  pfXformPt3(vec, vec, FTR);
  x = vec[PF_X]; y = vec[PF_Y]; z = vec[PF_Z];

}
```

```
///////////////////////////////////////////////////////////////////////////////
//
///////////////////////////////////////////////////////////////////////////////
double Round_Earth_Utils::alt_msl(double& x, double& y, double& z)
{
  double altitude;
  pfVec3 location;

  if (!flat_earth)
    {
     pfSetVec3(location, x, y, z);
     altitude = pfDistancePt3(location, flat_earth_center) - local_origin_radius;
    }
   else
     altitude = z;
  return altitude;

}

///////////////////////////////////////////////////////////////////////////////
//
///////////////////////////////////////////////////////////////////////////////
void Round_Earth_Utils::ECI_to_ECEF(double& x, double& y, double& z,
                                    int seconds)
{
  pfMatrix z_rotation;
  double percent_rotation, angle_rotated;
  pfVec3 sat_pos;

  pfSetVec3(sat_pos, x, y, z);
  percent_rotation = ( (float)seconds )/( (float)SECONDS_PER_DAY );
  angle_rotated = -360.0*percent_rotation;
  pfMakeRotMat(z_rotation, angle_rotated, 0.0, 0.0, 1.0);
  pfXformPt3(sat_pos, sat_pos, z_rotation);
  x = sat_pos[PF_X]; y = sat_pos[PF_Y]; z = sat_pos[PF_Z];

}

///////////////////////////////////////////////////////////////////////////////
//
///////////////////////////////////////////////////////////////////////////////
void Round_Earth_Utils::ECEF_to_ECI(double& x, double& y, double& z,
                                                  int seconds)
{
  pfMatrix z_rotation;
  double percent_rotation, angle_rotated;
  pfVec3 sat_pos;

  pfSetVec3(sat_pos, x, y, z);
  percent_rotation = ( (float)seconds )/( (float)SECONDS_PER_DAY );
  angle_rotated = 360.0*percent_rotation;
  pfMakeRotMat(z_rotation, angle_rotated, 0.0, 0.0, 1.0);
  pfXformPt3(sat_pos, sat_pos, z_rotation);
  x = sat_pos[PF_X]; y = sat_pos[PF_Y]; z = sat_pos[PF_Z];

}

//////////////////////////////////////////////////////////////////////////////
//
//////////////////////////////////////////////////////////////////////////////
void Round_Earth_Utils::reference_frame_mod_ftr(double z_rot, double y_rot,
                                                double x_rot)
{
  pfMatrix z, y, x;

  pfMakeRotMat(z, z_rot, 0.0, 0.0, 1.0);
  pfMakeRotMat(y, y_rot, 0.0, 1.0, 0.0);
  pfMakeRotMat(x, x_rot, 1.0, 0.0, 0.0);
  pfMultMat(ftr_twist, z, y);
  pfMultMat(ftr_twist, ftr_twist, x);

}

//////////////////////////////////////////////////////////////////////////////
//
//////////////////////////////////////////////////////////////////////////////
void Round_Earth_Utils::reference_frame_mod_rtf(double z_rot, double y_rot,
                                                double x_rot)
{
  pfMatrix z, y, x;

  pfMakeRotMat(z, z_rot, 0.0, 0.0, 1.0);
  pfMakeRotMat(y, y_rot, 0.0, 1.0, 0.0);
  pfMakeRotMat(x, x_rot, 1.0, 0.0, 0.0);
  pfMultMat(rtf_twist, z, y);
  pfMultMat(rtf_twist, rtf_twist, x);

}

//////////////////////////////////////////////////////////////////////////////
//
//////////////////////////////////////////////////////////////////////////////
pfSeg Round_Earth_Utils::core_segment(double x, double y, double z)
{
  pfSeg  segment;
  pfSeg  *seg_ptr;
  pfVec3 my_pos;

  if (!flat_earth)
     {
      seg_ptr = &segment;
      pfSetVec3(my_pos, x, y, z);
      pfMakePtsSeg(seg_ptr, my_pos, flat_earth_center);
     }
  else
     {
      PFSET_VEC3(segment.dir, 0.0f, 0.0f, -1.0f);
      segment.length = z;
     }
  return segment;

}

//////////////////////////////////////////////////////////////////////////////
//
//////////////////////////////////////////////////////////////////////////////
void Round_Earth_Utils::lat_long_from_xyz(double x, double y, double z,
  double orig_x, double orig_y, double orig_z,
  float& Latitude, float& Longitude)
{
    static int computed = FALSE;
    double pos_x, pos_y, pos_z;
    pfMatrix rtf,ftr;
    pfVec2  XYProj;
    float   Distance;
    pos_x = x;
    pos_y = y;
    pos_z = z;
//    cout << "Old Pos (XYZ): " << x << ", " << y << ", " << z ;


    if (!computed)
       {
        round_flat_xforms_2(orig_x, orig_y, orig_z, rtf, ftr);
round_flat_xforms(orig_x, orig_y, orig_z, rtf, ftr);
        computed = TRUE;
       }

    if (!flat_earth)
       {
        flat_pos_to_rnd(pos_x,pos_y,pos_z);
       }
    else
       {
        flat_pos_to_rnd_2(pos_x,pos_y,pos_z);
       }

//    cout << "New Pos (XYZ): " << pos_x << ", " << pos_y << ", " << pos_z ;

    /* Compute a latitude, and longitude from the position */
    PFSET_VEC2(XYProj, pos_x, pos_y);
    Distance = PFLENGTH_VEC2(XYProj);
    cout << "  Distance: " << Distance << endl;

    if (pos_x == 0.0f)
    {
if (pos_y < 0.0f)
{
    Longitude = -90.0f;
}
else
{
    Longitude = 90.0f;
}
    }
    else
    {
```

```
Longitude =atan2f(pos_y, pos_x) * RAD_TO_DEGR;
    }

    // Longitude is now +/- 180 degrees about z from the x axis

    if (Distance == 0.0f)
    {
if (pos_z > 0)
{
    Latitude = 90.0f;
}
else
{
    Latitude = -90.0f;
}
    }
    else
    {
Latitude = atan2f(pos_z,  Distance) * RAD_TO_DEGR;
    }
}

void Round_Earth_Utils::round_flat_xforms_2(double x, double y, double z,
  pfMatrix rtf, pfMatrix ftr)
{

  pfVec3    cur_pos;
  pfVec2    temp;
  double    theta, beta, sintheta, sinbeta, xyzplus2, xyplus2;
  pfMatrix  x_rot, z_rot, pfz_rot, pfy_rot, pfx_rot, trans_rtf, trans_ftr;


//  cout << "rfx called"<<endl;

  PFSET_VEC3(cur_pos, x, y, z);

  PFSET_VEC2(temp, x, y);

  xyzplus2 = PFLENGTH_VEC3(cur_pos);

  xyplus2  = PFLENGTH_VEC2(temp);
```

```
// Convert HPR of attached round earth player into Performer space

sintheta = cur_pos[PF_X]/xyplus2;

sinbeta  = xyplus2/xyzplus2;

theta    = pfArcSin(sintheta);  //angle needed for rotation around Z axis

beta     = pfArcSin(sinbeta);   //angle needed for rotation around X axis

// determine which quadrant angle is in and then determine by right hand rule
// how much of a rotation around the desired axis is needed

if ( x > 0.0f )                  // X-Y plane rotation around Z
{
  if ( y > 0.0f )       // quadrant 1
    theta = -(180.0f - theta);
  else   // quadrant 4
    theta = -theta;
}
else
{
  if ( y > 0.0f )     // quadrant 2
    theta = 180.0f + theta;
  else   // quadrant 3
    theta = -theta;
}

if ( z > 0.0f )       // Y-Z plane rotation around X
  beta = -beta;                           // quadrant 2
else
  beta = -(180.0f - beta);   // quadrant 3


pfMakeRotMat(z_rot, theta, 0.0f, 0.0f, 1.0f);  //heading

pfMakeRotMat(x_rot, beta,  1.0f, 0.0f, 0.0f);  //pitch

pfMultMat(rtf, z_rot, x_rot);  //this is the required order of mult for the terr

//make transpose before inserting translations so proper translation occurs
```

```
  pfTransposeMat(ftr, rtf);

  pfCopyMat(FTR_2, ftr);

  pfCopyMat(RTF_2, rtf);

  rtf[3][2] = -xyzplus2;

  ftr[3][0] = x;
  ftr[3][1] = y;
  ftr[3][2] = z;

  pfCopyMat(ftr_trans_2, ftr);

  pfCopyMat(rtf_trans_2, rtf);

}

void Round_Earth_Utils::flat_pos_to_rnd_2(double& x, double& y, double& z)
{
  pfVec3 pos;

  pfSetVec3(pos, x, y, z);
  pfXformPt3(pos, pos, ftr_trans_2);
  x = pos[PF_X]; y = pos[PF_Y]; z = pos[PF_Z];

}
```

Bibliography

1. Bailor, Paul. "Principles of Embedded Software." Course notes for CSCE693, Department of Electrical and Computer Engineering, Air Force Institute of Technology, October 1993.
2. Barnes, A and Lytham St Annes. "The Compromise Between Accuracy and Realism in Flight Simulation." *AIAA Flight Simulation Technologies Conference*. p. 65–71. 1991.
3. Bishop, G. and H. Fuchs. "Research Directions in Virtual Environments," *Computer Graphics*, *Volume 26*:p. 940–946 (August 1992).
4. Bouton, Frank. *Support Software Serial Data Protocol/Drivers*. Thrustmaster Incorporated, 10150 S.W. Nimbus Avenue, Suite E-7, Tigard, OR 97223, 1992.
5. Brunderman, John A. *Design And Application Of An Object Oriented Graphical Database Management System For Synthetic Environments*. MS thesis, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, AFIT/GCS/ENG/91D-01, December 1991.
6. Comeaux, J. *AF MANUAL 51-40, Flying Training Air Navigation*. Department Of The Air Force, Headquarters US Air Force, Washington DC 20330, 1993.
7. Cooke, Joseph M. *Parameterized Flight Dynamics Simulation System Using Quaternions*. MS thesis, Naval Post-Graduate School (NPS), Monterey, CA, March 1992.
8. Cruz-Neira, Carolina, et al. "Surround-Screen Projection-Based Virtual Reality: The Design and Implementation of the CAVE." *Computer Graphics Proceedings, Annual Conference Series*. p. 135–142. 1993. Proceedings of SIGGRAPH 93 (Anaheim, California, August 1-6, 1993).
9. Cruz-Neira, Carolina, et al. "The Cave Audio Visual Experience Automatic Virtual Environment," *Communications of the ACM*, *Volume 35*(no. 6):p. 65–72 (1992).
10. Cubic Defense Systems, AD/YIC, Eglin AFB, FL 32452-5000. *Computer Program Performance Specification For The Display And Debriefing Subsystem (DDS) Of The Red Flag Measurement And Debriefing System*, 1992.
11. Department Of The Air Force, Headquarters US Air Force, Washington DC 20330. *T.O. 1F-16A-34-1-1*, 1992.
12. Erichsen, Matthew N. *Weapon System Sensor Integration For A DIS-Compatible Virtual Cockpit*. MS thesis, Air Force Institute of Technology (AU). Wright-Patterson AFB OH, AFIT/GCS/ENG/93D-07, December 1993.
13. Foley, James D., et al. *Computer Graphics Principles And Practice* (Second Edition). Reading, Massachusetts: Addison-Wesley Publishing Company, 1990.

14. Ford Aerospace & Communications Corporation, Aeronutronic Division, Ford Road, Newport Beach, CA 92660. *Operator's Manual For Pave Tack F-111F*, 1981.

15. Gardner, Michael. *A Distributed Interactive Simulation Based Remote Debriefing Tool for Red Flag Missions*. MS thesis, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, AFIT/GCS/ENG/93D-09, December 1993.

16. Gerhard, William E. *Weapon System Integration For The AFIT Virtual Cockpit*. MS thesis, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, AFIT/GCS/ENG/93D-10, December 1993.

17. Haworth, Loren A. and Nancy M. Bucher. "Helmet-Mounted Dispay Systems for Flight Simulation." *SAE, Aerospace Technology Conference and Exposition, Anaheim, CA*. September 1989. SAE paper 892352.

18. Institute for Simulation and Training, 12424 Research Parkway, Suite 300, Orlando FL 32826. *Proposed IEEE Standard Draft Standard for Information Technology - Protocols for Distributed Interactive Simulation Applications Version 2.0 Second Draft*, March 1993. Contract Number N61339-91-C-0091.

19. Kacena, Neil. "The Six SSSSSS For Success," *USAF Fighter Weapons Review, Volume 33*(no. 4):p. 8–11 (Winter 1985). USAF FWS/COAR, Nellis AFB, NV.

20. Kunz, Andrea. *A Virtual Environment For Satellite Modeling And Orbital Analysis In A Distributed Interactive Simulation*. MS thesis, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, AFIT/GCS/ENG/93D-14, December 1993.

21. Lewantowicz, Zdzislaw H. "Fundamentals Of Aerospace Instruments and Navigation Systems." Course notes for EENG534, Department of Electrical and Computer Engineering, Air Force Institute of Technology, October 1989.

22. McLendon, Patricia. *Graphics Library Programming Guide*. Silicon Graphics Inc., Mountain View, California, 1991.

23. McLendon, Patricia. *IRIS Performer Programming Guide*. Silicon Graphics Inc., Mountain View, California, 1992.

24. Neyland, David. *The Zealous Pursuit Exercise Overview And Lessons Learned*. Defense Advanced Research Projects Agency, 1993.

25. Nicholas, Ted and Rita Rossi. *Military Cost Handbook* (Thirteenth Edition). Fountain Valley, CA: Data Search Associates, 1992.

26. Platt, Philip, et al. "Low-cost Approaches to Virtual Flight Simulation," *Proceedings of the IEEE National Aerospace and Electronics Conference, Volume 2*:p. 940–946 (1991).

27. Pope, Arthur R. *The SIMNET Network And Protocols*. Technical Report BBN Report No. 7627, 10 Moulton Street, Cambridge Massachusetts, 02138:

BBN Systems and Technologies, 1991. Prepared for Defense Advanced Research Projects Agency Information and Science Technology Office.

28. Quick, John R. "System Requirements for a High Gain Dome Display Surface." *Cockpit Displays and Visual Simulation*, edited by Harry Assenheim and Herbert Bell. p. 183–191. 1990. SPIE Vol. 1289.
29. Rastellini, M. M. *AF MANUAL 51-37, Flying Training Instrument Flying.* Department Of The Air Force, Headquarters US Air Force, Washington DC 20330, 1986.
30. Rebo, Robert K. and Phil Amburn. "A Helmet-Mounted Virtual Environment Display System." *Helmet-Mounted Displays*, edited by Jerome T. Carollo. p. 80–84. 1989. SPIE Vol. 1116.
31. Rheingold, H *Virtual Reality.* New York: Summit House, 1991.
32. Rolfe, J.M. and K.J. Staples, editors. *Flight Simulation.* New York, New York: Cambridge University Press, 1986.
33. Sheasby, Steven M. *Management Of SIMNET And DIS Entities In Synthetic Environments.* MS thesis, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, AFIT/GCS/ENG/92D-16, December 1992.
34. Silicon Graphics Inc., Mountain View, California. *Performer 1.2 Man Pages*, 1993. On line help for beta version Performer 1.2.
35. Simpson, Dennis J. *An Application Of The Object-Oriented Paradigm To A Flight Simulator.* MS thesis, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, AFIT/GCS/ENG/91D-22, December 1991.
36. Snodgrass, Mike. "F-16C Avionics Set Up," *USAF Fighter Weapons Review, Volume 36*(no. 4):p. 25–27 (Summer 1988). USAF FWS/COAR, Nellis AFB, NV.
37. Snyder, Mark. *OBJECTSIM - A Reusable DIS Simulation Framework.* MS thesis, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, AFIT/GCS/ENG/93D-20, December 1993.
38. Soltz, Brian. *Graphical Tools For Situational Awareness Assistance For Large Synthetic Battle Spaces.* MS thesis, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, AFIT/GCS/ENG/93D-21, December 1993.
39. Stiffler, Donald R. "Graduate Level Situational Awareness," *USAF Fighter Weapons Review, Volume 36*(no. 4):p. 15–20 (Summer 1988). USAF FWS/COAR, Nellis AFB, NV.
40. "Strawman Distributed Interactive Simulation Architecture Description Document Volume I: Summary Description," March 1992. Document number ADST/WDL/TR-92-003010.

41. Sutherland, Ivan. "The Ultimate Display." *Proceedings of the IFIP Congress*. p. 506–508. 1965.

42. Switzer, John C. *A Synthetic Environment Flight Simulator The AFIT Virtual Cockpit*. MS thesis, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, AFIT/GCS/ENG/92D-17, December 1992.

43. Thomas, Melvin L, et al. "The Display for Advanced Research and Training: An "Inexpensive" Answer To Tactical Simulation." *Large-Screen-Projection, Avionic, and Helmet-Mounted Displays*, edited by Harry Assenheim, et al. p. 65–75. 1991. SPIE Vol. 1456.

44. Wilson, Kirk G. *Synthetic Battle Bridge: Information Visualization and User Interface Design Applications in a Large Virtual Reality Environment*. MS thesis, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, AFIT/GCS/ENG/93D-26, December 1993.

Vita

Capt Erichsen was born in Coos Bay, Oregon 7 December 1959. He attended the United States Air Force Academy and graduated with a Bachelor of Science in 1982. Capt Erichsen attended Undergraduate Pilot Training and was an instructor in the T-37 for two consecutive assignments in the 85th FTS, 47th FTW Laughlin AFB, Texas and in the 559 FTS, 12 FTW Randolph AFB, Texas. In 1989 Capt Erichsen was assigned to the 494 TFS, 48 TFW, RAF Lakenheath, England and completed transition training in the F-111F. While in the 494^{th} he held positions as a Flight Commander and Chief of Training, accumulating 720 hours in the F-111. Capt Erichsen has a total of 2700 flying hours and 65 combat sorties resulting from participation in operations DESERT STORM and PROVIDE COMFORT. In 1992 Capt Erichsen was selected to attend the Air Force Institute of Technology for completion of a Master of Science in Computer Science. Capt Erichsen is now assigned to the faculty of the United States Air Force Academy.

Permanent address: 910 North Birch
Coquille, Oregon 97423

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE
December 1993

3. REPORT TYPE AND DATES COVERED
Master's Thesis

4. TITLE AND SUBTITLE
WEAPON SYSTEM SENSOR INTEGRATION
FOR A DIS-COMPATIBLE VIRTUAL COCKPIT

5. FUNDING NUMBERS

6. AUTHOR(S)
Matthew N. Erichsen, Capt, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GCS/ENG/93D-07

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
ARPA/ASTO
3701 North Fairfax Drive
Arlington, Va 22203

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This thesis continues the Virtual Cockpit (VC) research which investigates distributed interactive virtual flying environments. The VC v1.0 used the SIMNET protocols to only communicate position and orientation over a common network. A simple cockpit instrumentation configuration and limited head-up display (HUD) showed aircraft state. With VC v1.0 weapons or sensors could not interact in the simulation environment. The VC v2.0 transitions from the SIMNET protocol to a partial implementation of the Distributed Interactive Simulation (DIS) v2.0.3 protocol. Simulated radar and forward looking infra-red (FLIR) sensors were developed to aid operator detection and designation when employing various munition types. Simulated munition types include: radar or IR missiles, free-fall, laser guided, or electro-optic (EO) guided bombs, and a 20mm cannon. Virtual environments were created with CRT out-the-window presentations, color NTSC and monochrome high-resolution helmet mounted displays employing Polhemus head tracking sensors, and simultaneously five-channels on BARCO projectors. Target graphics systems included SGI workstations with Onyx processors using Reality Engines. Graphics rendering was accomplished with an AFIT developed object oriented simulation software package based on the SGI Performer 1.2 application development environment.

14. SUBJECT TERMS
Flight Simulators, Distributed Interactive Simulation, Synthetic Environments, Radar, FLIR, Computer Graphics

15. NUMBER OF PAGES
152

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT
UNCLASSIFIED

18. SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

19. SECURITY CLASSIFICATION OF ABSTRACT
UNCLASSIFIED

20. LIMITATION OF ABSTRACT
UL